Computers & Industrial Engineering 56 (2009) 216–223

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A discrete particle swarm optimization algorithm for scheduling parallel machines

Ali Husseinzadeh Kashan, Behrooz Karimi *

Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez Avenue, P.O. Box 15875-4413, Tehran, Iran

ARTICLE INFO

Article history:
Received 27 May 2007
Received in revised form 14 January 2008
Accepted 21 May 2008
Available online 28 May 2008

Keywords:
Scheduling
Parallel machines
Particle swarm optimization
Makespan

ABSTRACT

As a novel evolutionary technique, particle swarm optimization (*PSO*) has received increasing attention and wide applications in a variety of fields. To our knowledge this paper investigates the first application of *PSO* algorithm to tackle the parallel machines scheduling problem. Proposing equations analogous to those of the classical *PSO* equations, we present a discrete *PSO* algorithm (*DPSO*) to minimize makespan (C_{max}) criterion. We also investigate the effectiveness of *DPSO* algorithm through hybridizing it with an efficient local search heuristic. To verify the performance of *DPSO* algorithm and its hybridized version (*HDPSO*), comparisons are made through using a recently proposed simulated annealing algorithm for the problem, addressed in the literature, as a comparator algorithm. Computational results signify that the proposed *DPSO* algorithm is very competitive and can be rapidly guided when hybridizing with a local search heuristic.

1. Introduction

The formal description of the identical parallel machines scheduling problem is as follows: a set $J = \{J_1, J_2, \ldots, J_n\}$ of n independent jobs are available to be processed on a set $M = \{M_1, M_2, \ldots, M_m\}$ of m identical machines. Each job should be carried out on one of the machines, where the required time for processing job i by a machine is given by p_i. The subset of jobs assigned to machine M_j in a schedule is denoted by S_{M_j}. Once processing of a job starts, it must be completed without interruption. Furthermore, each machine can only process one job at a time, and there is no precedence relation between jobs. The paper considers the problem of optimal assignment of jobs to machines to minimize the time by which the last job leaves the system, i.e., the makespan criterion. A mixed integer programming formulation of the minimum makespan is as follows:

$$\min \quad y$$

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1 \quad i = 1, \ldots, n \tag{1}$$

$$y - \sum_{i=1}^{n} p_i x_{ij} \geqslant 0 \quad j = 1, \ldots, m \tag{2}$$

Where the optimal value of y is the optimal value of makespan, i.e., C^*_{max} and

$$x_{ij} = \begin{cases} 1 & \text{if job } i \text{ is assigned to machine } j \\ 0 & \text{otherwise} \end{cases}.$$

This problem is known to be NP-hard (Garey & Johnson, 1979). Thus, it is unlikely to obtain the optimal schedule through polynomial time-bounded algorithms. Over the years there has been a great deal of research to develop efficient approaches for the problem. As a member of a family of algorithms known as list-scheduling algorithms, the well-known longest processing time (*LPT*) rule of Graham (1969) has received extensive attention because in terms of performance guarantee it tends to perform better. Based on this rule we start with an empty schedule and iteratively put a non-scheduled job with longest processing time of all remaining jobs on to the machine with currently minimal workload. This method yields a schedule no worse than $\frac{C_{max}(LPT)}{C^*_{max}} \leqslant \frac{4}{3} - \frac{1}{3m}$, Where C_{max} (*LPT*) denotes the makespan received by the *LPT* algorithm. This performance guarantee is proven to be tight (Graham, 1969). Coffman, Garey, and Johnson (1978) proposed an algorithm entitled *MULTIFIT* that affords the relation between bin-packing and makespan problems. Although the performance guarantee for *MULTIFIT* algorithm is tighter than that of *LPT* algorithm, it does not follow that *MULTIFIT* algorithm will produce better makespan than *LPT* algorithm for any given problem. Fatemi Ghomi and Jolai Ghazvini (1998) proposed a pairwise interchange (*PI*) algorithm for the problem that is also applicable for scheduling non-identical parallel machines and also non-simultaneous job arrivals. With the idea that the variance of completion times of the last job on each machine in the presence of job preemption is zero, they tried to minimize sum of ranges of machine finish times instead of the makespan. Gupta and Ruiz-Torres (2001) proposed a heuristic named *LISTFIT* based on bin-packing problem and list scheduling that its worst-case performance bound is no worse than that of *MULTIFIT* algorithm. Their computational results showed that the heuristic outperforms the *LPT* algorithm, the *MULTIFIT* algorithm, and the *COMBINE* methods of Lee and Mas-

* Corresponding author. Tel.: +98 21 66413034; fax: +98 21 66413025.
E-mail address: B.Karimi@aut.ac.ir (B. Karimi).

0360-8352/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2008.05.007

sey (1988) that utilizes the result of *LPT* algorithm as an initial solution for the *MULTIFIT* algorithm. Lee, Wu, and Chen (2006) proposed a simulated annealing (*SA*) algorithm for the problem and evaluated its performance in comparison with *LISTFIT* and *PI* algorithms. They claimed that their approach outperforms both comparator algorithms for all experimental frameworks. Hence, in this research we use the *SA* algorithm as a base comparator algorithm to evaluate our results.

The reminder of the paper is organized as follows: In the following section, we give a brief introduction to *PSO* algorithm and its inspiration. Our analogous equations and the proposed *DPSO* algorithm are presented in Section 3. In this section we also investigate the hybridized version of *DPSO* i.e., *HDPSO* algorithm. Section 4 investigates the effectiveness of the proposed algorithms in comparison with *SA* algorithm through computations. The paper will be concluded in Section 5.

2. An introduction to *PSO*

Population based stochastic local search techniques are a relatively new paradigm in the field of optimization. There are several nature inspired techniques belonging to this family that use metaphors as guides in solving problems. The most famous members are genetic algorithms that use the metaphor of genetic and evolutionary principles of fitness selection for reproduction to search solution spaces. In a similar fashion the collective behavior of insect colonies, bird flocks, fish schools and other animal societies are the motivation for Swarm Intelligence that is "any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies (Bonabeau, Dorigo, & Theraulaz, 1999)". The Particle Swarm Optimization (*PSO*) method is a relatively new member of the Swarm Intelligence field for solving optimization problems.

PSO was proposed by Kennedy and Eberhard (1995) and is one of the latest evolutionary optimization techniques for optimizing continuous nonlinear functions. Its biological inspiration is based on the metaphor of social interaction and communication in a flock of birds or school of fishes. In these groups, there is a leader who guides the movement of the whole swarm. The movement of every individual is based on the leader and on his own knowledge. Since *PSO* is population-based and evolutionary in nature, the individuals (i.e. particles) in a *PSO* algorithm tend to follow the leader of the group, i.e. the one with the best performance. In general, it can be said that the model that *PSO* is inspired, assumes that the behavior of every particle is a compromise between its individual memory and a collective memory. The principles that govern *PSO* algorithm can be stated as follows:

1. Every particle k, a potential solution, in the swarm begins with a randomized position and randomized velocity. The position and velocity for particle k in the n-dimensional search space are represented by the vectors $X_k = (x_{k1}, x_{k2}, \ldots, x_{kn})$ and $V_k = (v_{k1}, v_{k2}, \ldots, v_{kn})$, respectively, where $x_{kd} (d = 1, \ldots, n)$ represents the location and $v_{kd} (d = 1, \ldots, n)$ represents the flying velocity of particle k in the dth dimension of the search space.
2. Every particle k knows its position and the value of the objective function for that position. It also remembers at which position $P_k^t = (P_{k1}^t, P_{k2}^t, \ldots, P_{kn}^t,)$ it has achieved its highest performance.
3. Every particle can generate a neighborhood from every position. Hence, it is also a member of some neighborhood of particles, and remembers which particle (given by the index g) has achieved the best overall position in that neighborhood. This neighborhood can either be a subset of the particles (local neighborhood), or all the particles (global neighborhood).
4. In each iteration t the behavior of particle is a compromise among three possible alternatives: following its current pattern of exploration; going back towards its best previous position; going back towards the best historic value of all particles.

This compromise is executed by the following equations at the current iteration of the algorithm:

$$v_{kd}^{t+1} = wv_{kd}^t + c_1 r_1 (p_{kd}^t - x_{kd}^t) + c_2 r_2 (p_{gd}^t - x_{kd}^t) \quad (3)$$

$$x_{kd}^{t+1} = x_{kd}^t + v_{kd}^{t+1} \quad (4)$$

where w, called the inertia weight, is a constant value chosen by the user to control the impact of the previous velocities on the current velocity. c_1 is the weight given to the attraction to the previous best location of the current particle and c_2 is the weight given to the attraction to the previous best location of the particle neighborhood. r_1 and r_2 are uniformly distributed random variables in $[0,1]$.

The original *PSO* algorithm can only optimize problems in which the elements of the solution are continuous real numbers. Because it is not possible to continuously "fly" particles through a discrete-valued space, a modification of the *PSO* algorithm for solving problems with binary-valued solution elements is developed by the initiators of *PSO* (Kennedy & Eberhard, 1997). Another approach for tackling discrete optimization problems by *PSO* also has been proposed by Laskari, Parsopoulos, and Vrahatis (2002) which is based on the truncation of the real values to their nearest integer. In recent years a considerable amount of efforts has been put on solving sequencing problems by *PSO* algorithm. One of the basic approaches in solving sequencing problems by *PSO* lies on representing a sequence of n jobs with an array of n real numbers (Fatih Tasgetiren, Liang, Sevkli, & Gencyilmaz, 2007; Liu, Wang, & Jin, 2008). The decoding process then is done based on some rules. For example the position of the smallest value in the array determines the job placed first in the permutation. Then the position of the second smallest value in the array determines the job placed second in the permutation and so on. Another approach to tackle discrete optimization with *PSO* is done by generating equations similar to the original *PSO* equations for updating the particle's position and velocity vectors (Pan, FatihTasgetiren, & Liang, 2008; Lian, Jiao, & Gu, 2006). However in these cases the proposed equations are not completely similar to the original ones.

In this paper we introduce analogous structure to the classical *PSO* equations, which enables to maintain all major characteristics of *PSO*. Unlike those equations proposed in the above papers, ours are completely similar to the original *PSO* equations for sowing parallel machines scheduling problem.

3. The proposed discrete *PSO* algorithm

This section describes how *PSO* meta-heuristic is adapted to solve the parallel machines makespan problem. One of the key issues when designing the *PSO* algorithm lies in its solution representation where particles bear the necessary information related to the problem domain on hand. In order to construct a direct relationship between the domain of the parallel machines scheduling problem and the *PSO* particles, n numbers of dimensions are presented each for one of n jobs. Hence, a solution for the problem of assigning jobs to machines is represented by an array whose length is equal to the number of jobs. The location of a particle

Job1	Job2	Job3	Job4	Job5
1	2	2	1	1

Fig. 1. Illustration of a particle's solution.

in the dth dimension represents the machine to which the dth job is assigned. Fig. 1 shows a particle representation for a scheduling problem with five jobs and two machines.

The process of generating a new position for a selected individual in the swarm is depicted in the following equations:

$$V_k^{t+1} = V_k^t \overset{+}{\mathrm{O}} \left(\left(R_1 \overset{\times}{\mathrm{O}} \left(P_k^t \bar{\mathrm{O}} X_k^t \right) \right) \overset{+}{\mathrm{O}} \left(R_2 \overset{\times}{\mathrm{O}} \left(P_g^t \bar{\mathrm{O}} X_k^t \right) \right) \right) \tag{5}$$

$$X_k^{t+1} = X_k^t \overset{+}{\mathrm{O}} V_k^{t+1} \tag{6}$$

where R_1 and R_2 are 1-by-n arrays comprising 0 or 1 elements. These random arrays are generated form a Bernoulli distribution in which the probability of getting 1 is equal to 0.3 (this value found suitable through our preliminary computations). However the suitable value always should be selected via tuning process considering the problem data. V_k^t and X_k^t are the kth particle current velocity and position arrays, respectively. P_k^t and P_g^t are the kth particle best position and the global best position visited so far. The definitions of the operators, used in the body of (5) and (6) are as follow.

The subtract operator ($\bar{\mathrm{O}}$). Differences between the current position of the kth particle, X_k^t, and a desired position P_k^t (or P_g^t) can be presented by an array of n elements in which, each element shows that whether the content of the corresponding element in X_k^t is different from the desired one or not. If yes, that element gets its value from P_k^t (or P_g^t). For those elements that have the same content in X_k^t and P_k^t (or P_g^t), their corresponding jobs are listed based on *LPT* order and are assigned to machines successively, whenever a machine becomes free. Fig. 2 illustrates the manner in which $\bar{\mathrm{O}}$ operator performs. More precisely, the number of elements that have not the same value in both X_k^t and P_k^t (or P_g^t) are equal to the Hamming distance between X_k^t and P_k^t (or P_g^t). Worth to mention that, when X_k^t is exactly equal to P_k^t or P_g^t, we omit the terms $R_1 \overset{\times}{\mathrm{O}} (P_k^t \bar{\mathrm{O}} X_k^t)$ or $R_2 \overset{\times}{\mathrm{O}} (P_g^t \bar{\mathrm{O}} X_k^t)$, since they are null arrays.

The multiply operator ($\overset{\times}{\mathrm{O}}$). By this operator we can add exploration ability to *DPSO* system. This ability is added by generating different binary vectors for two difference vectors and doing multiplication process. These random binary arrays perform the random numbers task in real-valued *PSO*. The $\overset{\times}{\mathrm{O}}$ operator is equivalent to Hadamard product. The Hadamard product of two 1-by-n arrays A and B is denoted by $A \cdot B$ and is an 1-by-n array given by $(A \cdot B)_i = a_i b_i$. Fig. 3 illustrates the manner in which $\overset{\times}{\mathrm{O}}$ operator performs.

As an alternative strategy, one my apply $\overset{\times}{\mathrm{O}}$ operator on the result of $\bar{\mathrm{O}}$ operator before making it a complete solution using *LPT* rule.

The add operator ($\overset{+}{\mathrm{O}}$). This operator is a crossover operator that typically is used in Genetic algorithms. Crossover is viewed as the operator that has the mission of interchanging structural information developed during the search. For crossover operator, we select two positions (cut points) from particle chain randomly, and exchange dimensions between these two positions. This type of crossover is traditionally known as two-point crossover. Since, this type of crossover produces two new chains, we select one of them randomly as the result of add operator. Fig. 4 illustrates the manner in which $\overset{+}{\mathrm{O}}$ operator performs.

As it seems, the proposed equations have all major characteristics of the classical *PSO* equations. Only the inertia weight is a vector of ones. The following pseudo-code describes in detail the steps of *DPSO* algorithm. We use $C_{\max}(X)$ to denote the makespan value corresponding to X.

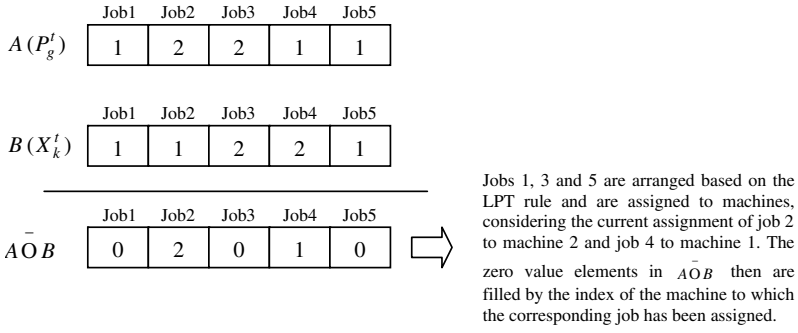

Fig. 2. Illustration of the manner in which $\bar{\mathrm{O}}$ operator performs.

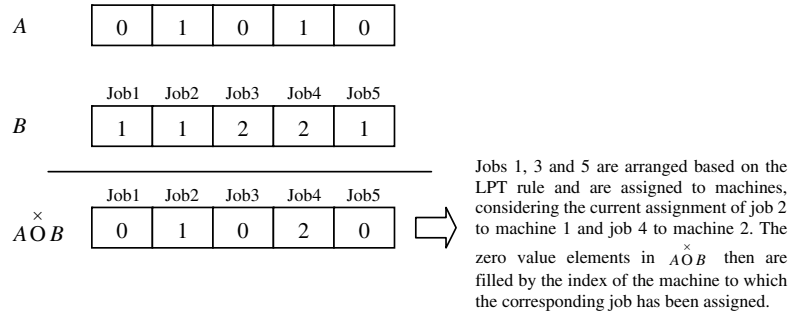

Fig. 3. Illustration of the manner in which $\overset{\times}{\mathrm{O}}$ operator performs.

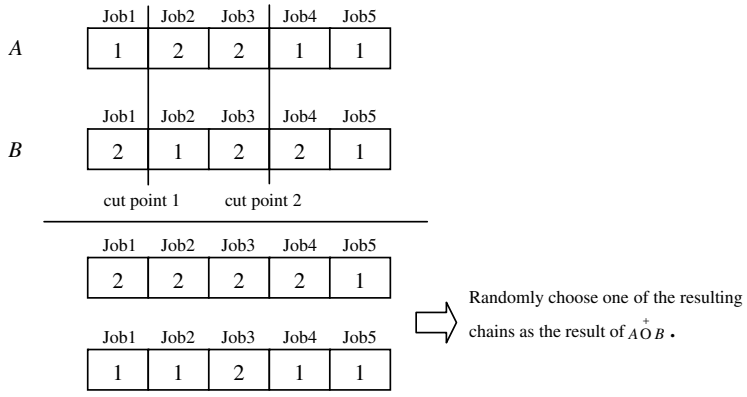

Fig. 4. Illustration of the manner in which $\overset{+}{O}$ operator performs.

Begin
$t = 0$;
For k: 1 *to swarm size*
  $X_k^t \leftarrow$*Generate a particle at random;*
  $P_k^t \leftarrow X_k^t$;
End For
$P_g^t \leftarrow \{X_l^t | \ l = \arg\min_{\forall k} \{C_{\max}(X_k^t)\}\}$;
Do
  For k: 1 *to swarm size*
    $V_k^{t+1} \leftarrow$*Update the* k^{th} *particle velocity vector using* (5);
    $X_k^{t+1} \leftarrow$*Update the* k^{th} *particle position vector using* (6);
    If $C_{\max}(X_k^{t+1}) < C_{\max}(P_k^t)$
      $P_k^{t+1} \leftarrow X_k^{t+1}$;
    Else
      $P_k^{t+1} \leftarrow P_k^t$;
    End If
  End For
  If $C_{\max}(P_g^t) > \min\{C_{\max}(P_k^{t+1})\}$
    $P_g^{t+1} \leftarrow \{P_l^{t+1} | \ l = \arg\min_{\forall k} \{C_{\max}(P_k^{t+1})\}\}$;
  End If
  $t \leftarrow t+1$;
Until *stopping criteria are true.*
End

To improve the effectiveness of *DPSO* algorithm, we hybridized it with a modified efficient local search algorithm, borrowed from our past research on scheduling identical parallel batch processing machines (Husseinzadeh Kashan, Karimi, & Jenabi, 2008). For a given schedule, the local search algorithm tries to reduce the makespan through suitable pairwise interchange of jobs between machines. The detail description of the local search algorithm applied on a given particle (schedule) is as follows:

Begin
1. Through inserting dummy jobs (with zero processing times) level the number of jobs assigned to each machine.
2. Arrange the machines in decreasing order of their finish time (FT), where the finish time is defined by the time in which the processing on the last job by the machine is completed;
3. If *all machines have the same finish time*
    Then stop; the optimal makespan has been achieved.
    Else $i \leftarrow 1; j \leftarrow m$; *and* $k \leftarrow 0$;
  End If
4. Select the machine M_i*. Considering the machine* M_j*, find a job* $a \in S_{M_i}$ *and a job (or dummy job)* $b \in S_{M_j}$ *such that:* $0 < p_a - p_b < FT_i - FT_j$.
5. If *there is not any pair of a and b*
    Then $j \leftarrow j - 1$; *and* $k \leftarrow k + 1$;
6. If $k = m - i$
    Then $i \leftarrow i + 1$; $j \leftarrow m$; *and* $k \leftarrow 0$;
7. If $i = m$
    Then *stop; No more improvement in* C_{max} *of the schedule is possible.*
    Else *go to 4;*
  End If
 Else *go to 4,*
 End If
Else *Interchange jobs a and b between* M_i *and* M_j*, then go to 2;*
End If
End

Worth to mention that the above local search algorithm is different with the original one in its first step. By inserting dummy jobs it is also possible, in an iteration of the algorithm, to pick a job from the queue of one machine and assign it to another machine. While in the original algorithm it was only possible to interchange a pair of jobs between the two machines.

For both *DPSO* and *HDPSO* algorithms we chose a swarm size (the number of particles generated in each iteration) of 10 particles. Both algorithms are stopped either after running for 100 iterations or when getting the lower bound value (*LB*). A lower bound on the optimal value of makespan is obtained when we relax the assumption stating: "Once a job begins processing, it must be completed without interruption". For this relaxed problem, McNaughton (1959) gives a formula for the minimum makespan that is $LB = \max\{\sum_{i=1}^{n} p_i/n, \max_{i=1,\ldots,n}\{p_i\}\}$. Clearly when all job processing times are integer, $\lceil LB \rceil$ is also a lower bound value, where $\lceil x \rceil$ denotes to the smallest integer larger than or equal to x.

Also, in the case of *HDPSO* algorithm every generated particle is selected to be improved by the local search algorithm with the probability of 0.1.

4. Computational experiments

In this section, we investigate a comparison study on the effectiveness of the *SA* algorithm of Lee et al. (2006) and both *PSO* and *HDPSO* algorithms by solving a large number of problems. We benchmark the experimental frameworks used in Lee et al. (2006) that originally have been designed by Gupta and Ruiz-Torres (2001), Lee and Massey (1988) and Kedia (1971). Four experimental frameworks, namely E_1, E_2, E_3 and E_4 are considered each of them having three influencing variables: the number of machines (m); the number of jobs (n); and the type of discrete uniform distribution used to generate job-processing times (p). Table 1 presents a summary of all experimental frameworks.

For experimental framework E_1, the number of machines (m) is set at three levels: 3, 4, and 5. The number of jobs (n) is set at three

Table 1
Summary of the computational experiments

	m	n	P
E_1	3, 4, 5	$2m$, $3m$, $5m$	$U(1,20)$, $U(20,50)$
E_2	2, 3, 4, 6, 8, 10	10, 30, 50, 100	$U(100,800)$
E_3	3, 5, 8, 10	$3m+1$, $3m+2$, $4m+1$, $4m+2$, $5m+1$, $5m+2$	$U(1,100)$, $U(100,200)$, $U(100,800)$
E_4	2	9	$U(1,20)$, $U(20,50)$, $U(50,100)$, $U(100,200)$, $U(100,800)$
	3	10	

levels: $2m$, $3m$, and $5m$. The processing times are drawn from discrete uniform ($U(.,.)$) distribution given at two levels, namely, $U(1,20)$ and $U(20,50)$. The experimental framework E_3 also can be interpreted in the same way. However the three influencing variables in E_3 have different configuration in comparison with E_1. For the experimental framework E_2, the number of machines is set at six levels: 2, 3, 4, 6, 8 and 10. The number of jobs is set at four levels: 10, 30, 50 and 100 (when n is equal to 10, m is only considered at 2 and 3). Here The job processing times are only generated from $U(100,800)$. For the experimental framework E_4, m and n are fixed, while the type of the uniform distribution in which the processing times are drown, is investigated at the combined five levels used in the three previous experiments. As the result, 120 experimental conditions are conducted. Each condition is replicated 50 times with different data. This yields in total 6000 problems.

All the algorithms are coded in MATLAB 6.5.1 and executed on a Pentium 4 CPU 2.66 GHz with 512 MB of RAM. Tables 2–7 report the comparison results obtained from the computations. The effectiveness of each algorithm is evaluated by two measures, i.e. the performance and the required execution time. Each cell belonging to the column with the caption "*Mean*" in the resulting tables, reports the mean performance obtained by the corresponding algorithm. The mean performance of an algorithm for an experiment is the average of ratios obtained in 50 times running of the algorithm on the experiment. Each ratio is given by dividing the reported value of makespan by the algorithm on the amount of lower bound, i.e. *LB*. It should be mentioned that the mean performance values are given by 4 number of digit precision, without rounding. Each cell belonging to the column with the caption "*Avg. Time*" in the resulting tables, reports the average time (in second) taken by the corresponding algorithm to solve the problems generated for the experiment.

Each of the last three columns in the resulting tables gives an indication on how many times a given algorithm reports a better makespan comparing with respect to the other algorithm. For instance, a value of *c/d* in column *SA/DPSO* means that, out of 50 problems, there are *c* problems for which *SA* yields a better solution than *DPSO*, *d* problems for which *DPSO* performs better, and 50-*c*-*d* problems for which *SA* and *DPSO* yield the same makespan.

The results for the experimental framework E_1 are summarized in Table 2. In terms of the mean performance there is not a significant difference between *DPSO* and *HDPSO* algorithms. However, there are three experiments for which *HDPSO* algorithm performs slightly better than *DPSO* algorithm. Although in most of the experiments the performance of *SA* is quite similar to that of *DPSO* and *HDPSO*, there are eight experiments for which both *DPSO* and *HDPSO* algorithms perform better than *SA*. Also, as the number of jobs increases, at a fixed level of m, the mean performance of algorithms improves. In terms of the execution time, *SA* algorithm takes less time than both of *DPSO* and*HDPSO* algorithms in most of the experiments. However there are some experiments in which the time taken by *DPSO* or*HDPSO* algorithms is less than that of needed for *SA*. Also, there is not a meaningful difference between the two latter.

Table 3 reports the results on experimental framework E_2. Here the superiority of *HDPSO* over *SA* and *DPSO* algorithms is more pronounced in terms of the mean performance. Comparing *SA* with *DPSO* algorithm, there are some experiments which *DPSO* performs better than*SA* and vice versa. Tracking the trends in the mean performance of the algorithms reveals that for a given number of jobs, as the number of machines increases, the mean performance diminishes. Comparing the execution times, as the number of jobs increases, the time taken by the algorithms becomes longer. Also, increasing the number of machines adds to the running times. Although for the problems with 100 jobs and larger values of m, the mean performance of *SA* is a bit better than *DPSO*, it is significantly at the expense of the execution times. From the results it can be found that as the number of jobs increases, *HDPSO* algorithm is also the dominant algorithm in terms of the required execution time. For example, for the experiment with 100 jobs and six machines, while *SA* requires 32.1 s of CPU time, *HDPSO* algorithm needs only 0.49 s to report the optimal solutions. This is about 11.4 s for *DPSO*. One may found that as the number of jobs and

Table 2
Results for experiment E_1

m	n	p	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
			Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
3	6	$U(1,20)$	1.0592	0.06	1.0592	0.89	1.0592	0.92	0/0	0/0	0/0
	9		1.0089	0.05	1.0089	0.55	1.0089	0.53	0/0	0/0	0/0
	15		1.0000	0.01	1.0000	0.02	1.0000	0.02	0/0	0/0	0/0
	6	$U(20,50)$	1.0567	0.07	1.0567	1.17	1.0567	1.23	0/0	0/0	0/0
	9		1.0084	0.09	1.0084	1.09	1.0084	1.12	0/0	0/0	0/0
	15		1.0007	0.10	1.0001	0.13	1.0001	0.09	0/5	0/5	0/0
4	8	$U(1,20)$	1.0687	0.08	1.0687	1.56	1.0687	1.53	0/0	0/0	0/0
	12		1.0036	0.04	1.0022	0.13	1.0022	0.13	0/2	0/2	0/0
	20		1.0000	0.06	1.0000	0.03	1.0000	0.02	0/0	0/0	0/0
	8	$U(20,50)$	1.0597	0.1	1.0597	1.89	1.0597	1.80	0/0	0/0	0/0
	12		1.0110	0.18	1.0094	1.42	1.0088	1.50	2/12	0/12	0/2
	20		1.0010	0.20	1.0000	0.08	1.0000	0.04	0/9	0/9	0/0
5	10	$U(1,20)$	1.0587	0.10	1.0587	1.54	1.0587	1.73	0/0	0/0	0/0
	15		1.0101	0.11	1.0079	0.45	1.0079	0.49	0/4	0/4	0/0
	25		1.0000	0.10	1.0000	0.03	1.0000	0.03	0/0	0/0	0/0
	10	$U(20,50)$	1.0589	0.13	1.0586	2.07	1.0586	2.28	0/1	0/1	0/0
	15		1.0091	0.25	1.0071	1.46	1.0069	1.59	1/9	0/9	0/1
	25		1.0011	0.30	1.0001	0.24	1.0000	0.04	0/9	0/10	0/1
Average			1.0231	0.11	1.0225	0.81	1.0224	0.83	0.2/2.8	0/2.9	0/0.2

Table 3
Results for experiment E_2

m	n	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
		Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
2	10	1.0009	0.15	1.0008	1.25	1.0008	1.37	0/3	0/3	0/0
3		1.0071	0.21	1.0063	1.97	1.0063	2.17	0/11	0/11	0/0
2	30	1.0000	0.21	1.0000	0.07	1.0000	0.06	0/0	0/0	0/0
3		1.0002	1.53	1.0000	1.77	1.0000	0.32	1/32	0/38	0/13
4		1.0009	2.35	1.0004	3.11	1.0000	1.44	8/34	0/47	1/40
6		1.0027	2.13	1.0019	3.48	1.0004	4.81	15/29	0/48	0/48
8		1.0058	2.49	1.0050	3.64	1.0014	5.94	15/29	0/49	0/50
10		1.0109	2.08	1.0095	3.82	1.0035	6.75	19/28	0/48	0/50
2	50	1.0000	0.25	1.0000	0.09	1.0000	0.14	0/0	0/0	0/0
3		1.0001	4.11	1.0000	1.72	1.0000	0.18	1/32	0/37	0/6
4		1.0004	6.83	1.0001	4.18	1.0000	0.46	7/29	0/43	0/32
6		1.0014	7.89	1.0009	5.42	1.0000	3.07	14/33	1/48	0/50
8		1.0027	8.94	1.0020	5.72	1.0002	6.75	16/31	0/50	0/49
10		1.0031	7.91	1.0035	6.01	1.0005	10.17	33/13	0/50	0/50
2	100	1.0000	0.86	1.0000	0.1	1.0000	0.92	0/0	0/0	0/0
3		1.0000	12.19	1.0000	1.13	1.0000	0.44	0/9	0/9	0/0
4		1.0000	16.36	1.0000	6.79	1.0000	0.36	11/9	0/23	0/25
6		1.0002	32.10	1.0003	11.40	1.0000	0.49	25/12	0/46	0/49
8		1.0005	38.63	1.0007	12.19	1.0000	1.25	29/13	0/48	0/50
10		1.0000	40.01	1.0012	12.95	1.0000	3.65	45/2	0/46	0/50
Average		1.0018	9.36	1.0016	4.34	1.0006	2.53	11.9/17.4	0.1/32.1	0.1/28.0

Table 4
Results for experiment E_3: $p \sim U(1,100)$

m	n	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
		Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
3	10	1.0115	0.14	1.0107	1.37	1.0107	1.54	0/5	0/5	0/0
	11	1.0045	0.14	1.0037	0.92	1.0037	1.01	0/6	0/6	0/0
	13	1.0019	0.15	1.0005	0.37	1.0003	0.23	0/14	0/16	0/2
	14	1.0009	0.15	1.0000	0.14	1.0000	0.06	0/11	0/11	0/0
	16	1.0004	0.16	1.0000	0.09	1.0000	0.04	0/6	0/6	0/0
	17	1.0005	0.14	1.0000	0.04	1.0000	0.04	0/8	0/8	0/0
5	16	1.0096	0.39	1.0046	1.64	1.0039	1.76	0/27	0/29	0/6
	17	1.0086	0.5	1.0036	1.55	1.0024	1.45	1/29	0/35	0/11
	21	1.0031	0.52	1.0006	0.84	1.0000	0.28	1/25	0/29	0/7
	22	1.0030	0.50	1.0010	0.96	1.0000	0.25	0/20	0/28	0/12
	26	1.0020	0.60	1.0004	0.71	1.0000	0.13	1/20	0/25	0/6
	27	1.0010	0.57	1.0003	0.63	1.0000	0.10	0/8	0/13	0/5
8	25	1.0127	1.00	1.0094	2.97	1.0041	2.02	4/19	0/40	0/33
	26	1.0095	0.98	1.0069	3.01	1.0016	1.68	11/17	0/37	0/36
	33	1.0030	1.27	1.0028	2.83	1.0002	0.61	9/11	0/28	0/26
	34	1.0033	1.46	1.0027	2.81	1.0001	0.64	6/12	0/31	0/27
	41	1.0021	1.62	1.0012	2.54	1.0000	0.37	3/12	0/24	0/17
	42	1.0018	1.64	1.0013	2.39	1.0000	0.29	3/8	0/22	0/18
10	31	1.0123	1.42	1.0120	3.82	1.0050	3.41	13/12	0/35	0/36
	32	1.0094	1.38	1.0089	3.63	1.0011	2.40	12/14	0/41	0/40
	41	1.0039	2.15	1.0037	3.91	1.0004	1.24	12/12	0/30	0/31
	42	1.0045	2.58	1.0040	4.31	1.0005	1.59	10/15	0/34	0/34
	51	1.0030	3.14	1.0021	4.31	1.0000	0.53	6/15	0/35	0/27
	52	1.0026	3.08	1.0022	4.41	1.0000	0.37	5/9	0/30	0/28
Average		1.0047	1.07	1.0034	2.09	1.0014	0.91	4.0/13.9	0/24.9	0/16.7

the number of machines increase, *HDPSO* algorithm can report a better makespan for almost all of the 50 replications. Also, there is only one problem for *SA* and only one problem for *DPSO* algorithm (among 50×20 problems) that the performance of *HDPSO* algorithm is poorer in comparison with these algorithms.

Tables 4–6 present the results for experimental framework E_3 when the processing times are generated in $U(1,100)$, $U(100,200)$ and $U(100,800)$, respectively. As can be found, both *DPSO* and *HDPSO* algorithms reveal better performance than *SA* on all the experiments in each table. However, as it is expected, *DPSO* algorithm is also dominated by *HDPSO* algorithm. In Tables 4 and 5, results demonstrate that there is no problem instance (among 50×24 problems) for which *SA* or *DPSO* algorithms could report a better performance in comparison with *HDPSO* algorithm. However in Table 6 there are only two problem instances for *SA* and only 13 problem instances (among 50×24 problems) for *DPSO* algorithm that *HDPSO* algorithm performs poorer in comparison with these algorithms. In Tables 4 and 5, from the row with the caption "*Average*", it can be inferred that the average time taken by *HDPSO* algorithm is the shortest, while the average time needed for *DPSO* algorithm is the longest and the time taken by *SA* lies between the two extremes. This is due to the fact that, hybridizing with local search algorithm speeds up converging toward the global optimum. Hence, our second stopping condition, which is getting the lower bound solution, is fired soon and *HDPSO* algorithm stops. In Table 6, *SA* algorithm takes less time than *DPSO* and

Table 5
Results for experiment E_3: $p \sim U(100,200)$

m	n	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
		Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
3	10	1.0122	0.22	1.0119	2.07	1.0119	2.23	0/3	0/3	0/0
	11	1.0137	0.28	1.0129	2.08	1.0125	2.32	4/7	0/10	0/6
	13	1.0024	0.34	1.0011	1.56	1.0010	1.36	4/28	0/28	0/4
	14	1.0021	0.38	1.0014	1.43	1.0007	0.75	6/20	0/31	0/20
	16	1.0010	0.50	1.0002	0.77	1.0000	0.18	2/24	0/28	0/6
	17	1.0011	0.50	1.0002	0.89	1.0000	0.13	0/28	0/33	0/9
5	16	1.0130	0.63	1.0097	2.62	1.0087	3.18	2/27	0/35	0/17
	17	1.0141	0.80	1.0122	2.72	1.0096	3.35	9/22	0/37	0/27
	21	1.0043	1.07	1.0014	2.60	1.0000	0.62	1/35	0/45	0/39
	22	1.0042	1.22	1.0020	2.82	1.0002	0.98	10/35	0/44	0/39
	26	1.0035	1.39	1.0007	2.51	1.0000	0.30	0/44	0/49	0/30
	27	1.0027	1.82	1.0013	2.82	1.0000	0.35	6/31	0/46	0/38
8	25	1.0118	1.53	1.0071	3.36	1.0030	3.79	4/37	0/47	0/44
	26	1.0129	1.59	1.0077	3.44	1.0023	3.74	4/36	0/49	0/49
	33	1.0060	2.94	1.0027	3.77	1.0000	0.65	7/38	0/48	0/45
	34	1.0068	3.13	1.0032	3.97	1.0000	0.95	5/36	0/50	0/47
	41	1.0051	4.40	1.0019	4.29	1.0000	0.72	4/40	0/48	0/43
	42	1.0047	4.80	1.0022	4.65	1.0000	0.50	7/38	0/47	0/48
10	31	1.0134	2.34	1.0072	3.99	1.0015	3.58	4/41	0/50	0/48
	32	1.0137	2.76	1.0081	4.10	1.0016	4.06	9/40	0/50	0/48
	41	1.0091	4.60	1.0036	4.85	1.0002	1.54	2/43	0/50	0/50
	42	1.0085	4.86	1.0043	4.86	1.0000	1.03	4/40	0/50	0/49
	51	1.0057	8.36	1.0026	5.78	1.0000	0.54	2/41	0/50	0/50
	52	1.0065	8.05	1.0033	5.88	1.0000	0.97	3/40	0/50	0/50
Average		1.0074	2.43	1.0045	3.24	1.0022	1.57	4.1/32.2	0/40.7	0/33.5

Table 6
Results for experiment E_3: $p \sim U(100,800)$

m	n	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
		Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
3	10	1.0089	0.22	1.0078	2.12	1.0078	2.39	0/10	0/10	0/10
	11	1.0056	0.28	1.0047	2.11	1.0046	2.43	1/13	0/13	0/1
	13	1.0024	0.37	1.0013	2.20	1.0009	2.65	3/32	0/33	0/8
	14	1.0018	0.48	1.0006	1.96	1.0005	2.15	1/38	0/41	0/9
	16	1.0014	0.56	1.0003	2.00	1.0001	1.69	2/38	0/43	2/21
	17	1.0014	0.63	1.0002	1.77	1.0000	1.22	1/44	0/47	1/23
5	16	1.0102	0.56	1.0072	2.63	1.0057	3.43	5/35	0/42	1/23
	17	1.0088	0.71	1.0047	2.70	1.0032	3.53	3/39	0/48	0/32
	21	1.0046	1.19	1.0020	2.90	1.0006	3.71	6/41	0/47	3/42
	22	1.0046	1.20	1.0019	2.97	1.0005	3.69	7/41	0/49	0/44
	26	1.0035	1.76	1.0013	3.17	1.0003	3.53	10/37	0/49	3/42
	27	1.0029	1.94	1.0012	3.16	1.0002	3.82	10/37	0/49	2/44
8	25	1.0119	1.58	1.0087	3.26	1.0032	5.17	16/29	0/50	1/48
	26	1.0096	1.90	1.0076	3.33	1.0026	5.32	17/30	0/49	0/49
	33	1.0058	3.48	1.0043	3.88	1.0011	6.35	12/33	1/48	0/49
	34	1.0064	3.50	1.0040	4.08	1.0009	6.62	13/35	0/50	0/48
	41	1.0036	6.03	1.0026	4.76	1.0004	7.31	14/32	0/49	0/50
	42	1.0041	6.10	1.0025	4.87	1.0004	7.56	14/34	0/50	0/50
10	31	1.0125	2.51	1.0092	3.87	1.0032	7.03	19/30	0/48	0/49
	32	1.0118	2.99	1.0082	3.98	1.0026	7.21	12/34	1/49	0/50
	41	1.0059	5.84	1.0050	5.00	1.0009	8.93	20/25	0/49	0/50
	42	1.0066	6.81	1.0051	5.091	1.0010	8.89	20/30	0/50	0/49
	51	1.0046	9.27	1.0035	6.12	1.0004	10.44	18/25	0/50	0/50
	52	1.0042	9.89	1.0035	6.19	1.0004	10.38	16/28	0/50	0/50
Average		1.0059	2.90	1.0040	3.50	1.0017	5.22	10.0/32.0	0.1/44.2	0.5/37.1

HDPSO algorithms. Also the time needed for *DPSO* algorithm is less than *HDPSO* algorithm. Here the reason for the increment in the required running time of *HDPSO* algorithm is most probably related to the performance of the lower bound. Inevitably, *HDPSO* algorithm should pass 100 iterations to be stopped. Also the execution time of algorithms increases as the number of machines increases.

Table 7 present the results for experimental framework E_4. For the experiments in E_4 since the problem sizes are small, all of the algorithms perform almost similar. However, again *HDPSO* algorithm performs slightly better. In terms of the execution time this is the *SA* algorithm that performs better.

5. Conclusion

To our knowledge the first application of *PSO* algorithm in scheduling parallel machines to minimize makespan criterion is addressed in this paper. Replacing the classical +, − and × operators with their counterparts defined in this paper, we proposed

Table 7
Results for experiment E_4

m	n	t	SA		DPSO		HDPSO		SA/DPSO	SA/HDPSO	PSO/HDPSO
			Mean	Avg. time	Mean	Avg. time	Mean	Avg. time			
2	9	$U(1,20)$	1.0000	0.00	1.0000	0.01	1.0000	0.01	0/0	0/0	0/0
		$U(20,50)$	1.0009	0.03	1.0009	0.22	1.0009	0.21	0/0	0/0	0/0
		$U(1,100)$	1.0009	0.04	1.0009	0.25	1.0009	0.28	0/0	0/0	0/0
		$U(50,100)$	1.0038	0.11	1.0038	1.065	1.0038	1.07	0/0	0/0	0/0
		$U(100,200)$	1.0042	0.14	1.0040	1.22	1.0040	1.23	0/3	0/3	0/0
		$U(100,800)$	1.0017	0.14	1.0017	1.33	1.0017	1.46	0/0	0/0	0/0
3	10	$U(1,20)$	1.0014	0.02	1.0014	0.14	1.0014	0.14	0/0	0/0	0/0
		$U(20,50)$	1.0083	0.14	1.0071	1.16	1.0071	1.20	0/6	0/6	0/0
		$U(1,100)$	1.0095	0.14	1.0092	1.39	1.0092	1.50	0/2	0/2	0/0
		$U(50,100)$	1.0096	0.20	1.0092	1.61	1.0090	1.65	0/4	0/6	0/2
		$U(100,200)$	1.0167	0.22	1.0162	2.00	1.0162	2.08	0/5	0/5	0/0
		$U(100,800)$	1.0092	0.22	1.0092	1.95	1.0091	2.22	1/6	0/7	0/2
Average			1.0055	0.11	1.0053	1.02	1.0052	1.08	0.1/2.1	0/2.4	0/0.3

equations analogous to those of the classical *PSO* equations. In our discrete version of *PSO* (*DPSO*), the representations of the position and velocity of the particle is extended from the real vector to integer vector, by which we accomplished the mapping between the job scheduling problem and the particle. Through modifying an efficient local search algorithm taken from our previous research, we also presented the hybridized version of *DPSO* algorithm entitled *HDPSO* algorithm. Results obtained from extended computational efforts, justifies the competitive performance of *DPSO* algorithm comparing to a recently published benchmark algorithm, i.e., a simulated annealing (*SA*) approach. Also we found that *HDPSO* algorithm outperforms both *SA* and *DPSO* algorithms in all of the experimental frameworks.

For future research, extension of the proposed equations and applying the methodology to other equation based evolutionary algorithms such as Differential Evolution (*DE*) is attractive. Also, extending our approach for solving other performance criterion or even multi-objective parallel machines scheduling problems is interesting. Furthermore, using our approach in other cases of more complex machine environments, e.g. non-identical parallel machines, multi-stage flow-shop and job-shop problems, could be important from practical view point.

References

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: From natural to artificial systems. Santa Fe institute studies in the science of complexity*. New York, NY: Oxford University Press.

Coffman, E. G., Garey, M. R., & Johnson, D. S. (1978). An application of bin-packing to multi-processor scheduling. *SIAM Journal on Computing, 7*, 1–17.

Fatemi Ghomi, S. M. T., & Jolai Ghazvini, F. (1998). A pairwise interchange algorithm for parallel machine scheduling. *Production Planning & Control, 9*, 685–689.

Fatih Tasgetiren, M., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research, 177*, 1930–1947.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco, CA: Freeman.

Graham, R. L. (1969). Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics, 17*, 416–429.

Gupta, J. N. D., & Ruiz-Torres, A. J. (2001). A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning & Control, 12*, 28–36.

Husseinzadeh Kashan, A., Karimi, B., & Jenabi, M. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers & Operations Research, 35*, 1084–1098.

Kedia, S. K. (1971). *A job scheduling problem with parallel processors. Technical Report*. Ann Arbor, MI: Department of Industrial and Operations Engineering, University of Michigan.

Kennedy, J., & Eberhard, R. C. (1995). Particle swarm optimization. In: *Proceedings of IEEE International Conference on Neural Networks* (pp. 1942–1948). Piscataway, NJ, USA.

Kennedy, J., & Eberhard, R. C. (1997). A discrete binary version of the particle swarm algorithm. In: *Proceedings of IEEE Conference on Systems, Man, and Cybernetics* (pp. 4104-4109). Piscataway, NJ, USA.

Laskari, E. C., Parsopoulos, K. E., & Vrahatis, M. N. (2002). Particle swarm optimization for integer programming. In: *Proceedings of the IEEE 2002 Congress on Evolutionary Computation*, Honolulu (HI), pp. 1582–1587.

Lee, C. Y., & Massey, J. D. (1988). Multiprocessor scheduling: Combining LPT and MULTIFIT. *Discrete Applied Mathematics, 20*, 233–242.

Lee, W. C., Wu, C. C., & Chen, P. (2006). A simulated annealing approach to makespan minimization on identical parallel machines. *International Journal of Advanced Manufacturing Technology, 31*, 328–334.

Lian, Z., Jiao, B., & Gu, X. (2006). A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Applied Mathematics and Computation, 183*, 1008–1017.

Liu, B., Wang, L., & Jin, Y. H. (2008). An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 35*, 2791–2806.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science, 6*, 1–12.

Pan, Q. K., FatihTasgetiren, M., & Liang, Y. C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research, 35*, 2807–2839.