International Journal of Innovative Computing, Information and Control
Volume **6**, Number **9**, September **2010**

ICIC International ©2010 ISSN 1349-4198
pp. 1–09-0370

A DISCRETE PARTICLE SWARM OPTIMIZATION APPROACH FOR GRID JOB SCHEDULING

Hesam Izakian[1,2], Behrouz Tork Ladani[1], Ajith Abraham[2,*] and Václav Snášel[3]

[1]Department of Computer Engineering
University of Isfahan
Hezar Jerib Avenue, Isfahan, Iran
hesam.izakian@gmail.com; ladani@eng.ui.ac.ir

[2]Machine Intelligence Research Labs (MIR Labs)
Washington 98071, USA
http://www.mirlabs.org
*Corresponding author: ajith.abraham@ieee.org

[3]Faculty of Electrical Engineering and Computer Science
VSB-Technical University of Ostrava
Ostrava, Czech Republic
vaclav.snasel@vsb.cz

Received March 2009; revised September 2009

Abstract. *Scheduling is one of the core steps to efficiently exploit the capabilities of emergent computational systems such as grid. Grid environment is a dynamic, heterogeneous and unpredictable one sharing different services among many different users. Because of heterogeneous and dynamic nature of grid, the methods used in traditional systems could not be applied to grid scheduling and therefore new methods should be looked for. This paper represents a discrete Particle Swarm Optimization (DPSO) approach for grid job scheduling. PSO is a population-based search algorithm based on the simulation of the social behavior of bird flocking and fish schooling. Particles fly in problem search space to find optimal or near-optimal solutions. In this paper, the scheduler aims at minimizing makespan and flowtime simultaneously in grid environment. Experimental studies illustrate that the proposed method is more efficient and surpasses those of reported meta-heuristic algorithms for this problem.*
Keywords: Grid computing, Scheduling, Makespan, Flowtime, Particle swarm optimization

1. **Introduction.** Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and in some cases, high-performance orientation [1]. Grid is composed of a set of virtual organizations (VOs). Each VO has its various resources and services, and on the basis of its policies provides access to them and hence grid resources and services are much different and heterogeneous, and are distributed in different geographically areas. At any moment, different resources are added to or removed from grid, and as a result, grid environment is highly dynamic.

Grid resources are registered within one or more Grid Information Services (GISs). The end users submit their requests to the Grid Resource Broker (GRB). Different requests demand different requirements and available resources have different capabilities. GRB discovers proper resources for executing these requests by querying in GIS and then schedules them on the discovered resources.

According to the type of jobs being scheduled, the scheduling problem can be classified into two types: scheduling set of independent jobs and scheduling a directed acyclic graph (DAG) composed of communicating jobs. In this paper we consider the first type which involves allocation of a set of independent jobs from different users to a set of computing resources. This framework is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [18], factoring large numbers [37], and the Mersenne prime search [38]. These applications are most suited for computational grids, because communication can easily become a bottleneck for tightly coupled parallel applications. Condor [27] and APST [39] are one of the first projects providing specific support for such applications.

Since grid environments are very dynamic and the computing resources are very heterogeneous, the methods used in traditional systems could not be applied to grid job scheduling and therefore new methods should be looked for. An appropriate and optimal scheduling in grid results in more efficient use of available resources and hence higher satisfaction in users. Until now some works has been done in order to schedule jobs in grid. Yet according to new nature of the subject further research is required.

Cao [3] used agents for resource management and scheduling in grid environment. In this method different resources and services are regarded as different agents and grid resource discovery and advertisement are performed by these agents. Buyya [4] used economic based concepts including commodity market, posted price modeling, contract net models, bargaining modeling, etc for grid scheduling. He et al. [5], Cheng et al. [6] and Kumar et al. [7] used fuzzy methods for grid scheduling.

As mentioned in [8] scheduling is NP-complete. Meta-heuristic methods have been used to solve well-known NP-complete problems (e.g. [9,11,25,40]). Efficient Meta-heuristic methods, which are used frequently, are simulated annealing (SA) [23], genetic algorithm (GA) [24], ant colony optimization (ACO) [26] and particle swarm optimization (PSO) [17].

Yarkhan and Dongarra [10] used simulated annealing approach for grid job scheduling. Page and Naughton [13] used a genetic algorithm method for scheduling heterogeneous computing systems. In this method the scheduling strategy operates in a dynamically changing computing resource environment and adapts to variable communication costs and variable availability of processing resources.

Braun et al. [15] described eleven heuristics and compared them on different types of heterogeneous computing environments. The authors illustrated that the GA scheduler can obtain better results in comparison with others. Carretero et al. [29] used Genetic Algorithm-based schedulers for computational grids and most of GA operators are implemented and compared to find the best GA scheduler for this problem. In [35] the authors also focused on Struggle Genetic Algorithms and their tuning for scheduling of independent jobs in computational grids and hash-based implementations of the struggle Genetic operator for the GAs were proposed.

Izakian et al. [12] compared six efficient and popular pure heuristics for scheduling meta-tasks in heterogeneous computing systems. In [16] authors used a fuzzy particle swarm optimization for minimizing makespan and flowtime in computational grids. Ritchie and Levine [36] used a hybrid ant colony optimization for scheduling in HC systems. In this method, authors combined ant colony optimization with local and tabu search to find shorter schedules.

Different criteria can be used for evaluating efficacy of scheduling algorithms, the most important of which are makespan and flowtime. Makespan is the time when grid finishes the latest job and flowtime is the sum of finalization times of all the jobs. An optimal schedule will be the one that optimizes the flowtime and makespan [16,28,29]. The method

proposed in [16,28,29] aims at simultaneously minimizing make span and flowtime. In this paper, a version of discrete particle swarm optimization (DPSO) is proposed for grid job scheduling and the goal of scheduler is to minimize the two parameters mentioned above simultaneously. This method is compared to the methods presented in [15,16,25,36] in order to evaluate its efficacy. The experimental results show the presented method is more efficient than others and this method can be effectively used for grid scheduling. The remainder of this paper is organized in the following manner; in Section 2 we formulate the problem, in Section 3 PSO paradigm is briefly discussed, Section 4 describes our proposed method for grid job scheduling, and Section 5 reports the experimental results. Finally Section 6 concludes this work.

2. **Problem Formulation.** As mentioned in Section 1, GRB is responsible for scheduling by receiving the jobs from the users and querying their required services in GIS and then allocating these jobs to the discovered services. Suppose in a specific time interval, n jobs $\{J_1, J_2, ..., J_n\}$ are submitted to GRB. Also assume the jobs are independent of each other (with no inter-task data dependencies) and preemption is not allowed (they cannot change the resource they has been assigned to). At the time of receiving the jobs by GRB, m machines $\{M_1, M_2, ..., M_m\}$ are within the grid. In this paper scheduling is done at the machine level and it is assumed that each machine uses First-Come, First-Served (FCFS) method for performing the received jobs.

In this paper the expected time to compute (ETC) values of each job on each machine is assumed to be known based on user-supplied information, experiential data, job profiling and analytical benchmarking, or other techniques. Determination of ETC values is a separate research problem, and the assumption of such ETC information is a common practice in mapping research [2]. In [12,14,15,29,35] ECT matrices are used to estimate the required time for executing each job in each machine. An ETC matrix is an $n \times m$ matrix in which n is the number of jobs and m is the number of machines. One row of the ETC matrix contains the estimated execution time for a given job on each machine. Similarly one column of the ETC matrix consists of the estimated execution time of a given machine for each job. Thus, for an arbitrary job J_j and an arbitrary machine M_i, $\mathrm{ETC}(J_j, M_i)$ is the estimated execution time of J_j on M_i. In ETC model we take the usual assumption that we know the computing capacity of each resource, an estimation or prediction of the computational needs of each job, and the load of prior work of each resource.

Assume that E_{ij} ($i \in \{1, 2, ..., m\}$, $j \in \{1, 2, ..., n\}$) is the execution time for performing jth job in ith machine and W_i ($i \in \{1, 2, ..., m\}$) is the previous workload of M_i (the time required for performing the jobs given to it in the previous steps), then Equation (1) shows the time required for M_i to complete the jobs included in it. According to the aforementioned definition, makespan and flowtime can be estimated using Equation (2) and Equation (3) respectively.

$$\sum_{\forall\, job\, j\, allocated\, to\, machine\, i} E_{ij} + W_i \tag{1}$$

$$makespan = \max\{\sum_{\forall\, job\, j\, allocated\, to\, machine\, i} E_{ij} + W_i\}, \quad i \in \{1, 2, ..., m\} \tag{2}$$

$$flowtime = \sum\nolimits_{i=1}^{m} \sum_{\forall\, job\, j\, allocated\, to\, machine\, i} E_{ij} \tag{3}$$

As mentioned in the previous section the goal of the scheduler in this paper is to minimize makespan and flowtime simultaneously.

3. **Particle Swarm Optimization.** Particle Swarm Optimization (PSO) is a population based search algorithm inspired by bird flocking and fish schooling originally designed and introduced by Kennedy and Eberhart [17] in 1995. The basic PSO has been applied successfully to a number of problems including standard function optimization problems [30-32], solving permutation problems [33], and training multi-layer neural networks [20,21,32,34] and its use is rapidly increasing. A PSO algorithm contains a swarm of particles in which each particle includes a potential solution. In contrast to the evolutionary computation paradigms such as genetic algorithm, a *swarm* is similar to a population while a *particle* is similar to an individual. The particles fly through a multidimensional search space in which the position of each particle is adjusted according to its own experience and the experience of its neighbors. PSO system combines local search methods (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation [25].

3.1. **Binary PSO.** Primarily PSO was successfully used to solve continuous problems. In 1997 the binary version of this algorithm was presented by Kennedy and Eberhart [22] for discrete optimization problems. In this method each particle is composed of D elements which indicate a potential solution. In order to evaluate the appropriateness of solutions a fitness function is always used. Each particle is considered as a position in a D-dimensional space and each element of a particle position can take the binary value of 0 or 1 in which 1 means "included" and 0 means "not included". Each element can change from 0 to 1 and vise versa. Also each particle has a D-dimensional velocity vector the elements of which are in range $[-V_{\max}, V_{\max}]$. Velocities are defined in terms of probabilities that a bit will be in one state or the other. At the beginning of the algorithm, a number of particles and their velocity vectors are generated randomly. Then in some iteration the algorithm aims at obtaining the optimal or near-optimal solutions based on its predefined fitness function. The velocity vector is updated in each time step using two best positions, *pbest* and *nbest*, and then the position of the particles is updated using velocity vectors. *Pbest* and *nbest* are D-dimensional, the elements of which are composed of 0 and 1 the same as particles position and operate as the memory of the algorithm. The personal best position, *pbest*, is the best position the particle has visited and *nbest* is the best position the particle and its neighbors have visited since the first time step. Based on the size of neighborhoods two PSO algorithms can be developed. When all of the population size of the swarm is considered as the neighbor of a particle, *nbest* is called global best (*gbest*) and if the smaller neighborhoods are defined for each particle, then *nbest* is called local best (*lbest*). *gbest* uses the star neighborhood topology and *lbest* usually uses ring neighborhood topology. There are two main differences between *gbest* and *lbest* with respect to their convergence characteristics [32]. Due to the larger particle interconnectivity of the *gbest* PSO, it converges faster than the *lbest* PSO, but *lbest* PSO is less susceptible to being trapped in local optima. Equations (4) and (5) is used to update the velocity and position vectors of the particles respectively.

$$V_i^{(t+1)}(j) = V_i^t(j) + c_1 r_1 (pbest_i^t(j) - X_i^t(j)) + c_2 r_2 (nbest_i^t(j) - X_i^t(j)) \tag{4}$$

$$X_i^{(t+1)}(j) = \begin{cases} 1 & \text{if } sig(V_i^{(t+1)}(j)) > r_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where,

$$sig(V_i^{(t+1)}(j)) = \frac{1}{1 + \exp(-V_i^{(t+1)}(j))} \tag{6}$$

In Equation (4) $X_i^t(j)$ is jth element of ith particle in tth step of the algorithm and $V_i^t(j)$ is the jth element of the velocity vector of the ith particle in tth step. c_1 and c_2 are positive

acceleration constants which control the influence of *pbest* and *nbest* on the search process. Also r_1 and r_2 are random values in range [0, 1] sampled from a uniform distribution. r_{ij} in Equation (5) is a random number in range [0, 1] and Equation (6) shows sigmoid function. Figure 1 shows the pseudo-code of binary PSO. Stopping condition in Figure 1 can either be the maximum number of iterations, or finding an acceptable solution, or having no improvement in a number of iterations.

4. **The Proposed PSO Algorithm for Grid Job Scheduling.** In this section we propose a version of discrete particle swarm optimization for grid job scheduling. Particles need to be designed to present a sequence of jobs in available grid machines. Also the velocity has to be redefined. Details are given what follows.

```
Create and initialize a D-dimensional swarm with P particles
repeat
    for each particle i = 1, ..., P do
        if f(X_i) > f(pbest_i) then // f() represent the fitness function
            pbest_i = X_i;
        end
        if f(pbest_i) > f(nbest_i) then
            nbest_i = pbest_i;
        end
    end
    for each particle i = 1, ..., P do
      update the velocity vector using Equation (4)
      update the position vector using Equation (5)
    end
until stopping condition is true;
```

FIGURE 1. The pseudo-code of binary PSO

4.1. **Position of particles.** One of the key issues in designing a successful PSO algorithm is the representation step which aims at finding an appropriate mapping between problem solution and PSO particle. We use two types of representation, namely the direct and indirect representations. In direct representation solutions are encoded in a $1 \times n$ vector, called position vector, in which n is the number of jobs received by GRB in a time interval. The elements of this vector are natural numbers included in range [1, m], in which m is the number of available machines in the grid at the time of scheduling. Assume that X_k shows the position of kth particle; $X_k(j)$ indicates the machine where job j is assigned by the scheduler in this particle. Note that in this representation a machine number can appear more than once in a particle.

In indirect representation solutions are encoded in a $m \times n$ matrix, called position matrix, in which m is the number of available machines at the time of scheduling and n is the number of jobs. The position matrix of each particle has the two following properties:

1) All the elements of the matrices have either the value of 0 or 1. In other words if X_k is the position matrix of kth particles, then:

$$X_k(i,j) \in \{0,1\} \quad (\forall i,j), \quad i \in \{1,2,...,m\}, \quad j \in \{1,2,...,n\} \tag{7}$$

2) In each column of these matrices only one element is 1 and others are 0.

In position matrix each column represents a job allocation and each row represents allocated jobs in a machine. In each column it is determined that a job should be performed by which machine. Assume that X_k shows the position matrix of kth particle. If $X_k(i,j) = 1$ then the jth job will be performed by ith machine. Figures 2 and 3 show two equivalent potential solutions with direct and indirect representations in a problem with 5 jobs and 3 machines. These solutions show that J_2 and J_4 will be performed in M_1, J_3 and J_5 will be performed in M_2 and J_1 will be performed in M_3. As it could be realized from these figures the two presented methods for the problem encoding are easily convertible to each other.

J_1	J_2	J_3	J_4	J_5
3	1	2	1	2

FIGURE 2. Position vector (direct representation)

	J_1	J_2	J_3	J_4	J_5
M_1	0	1	0	1	0
M_2	0	0	1	0	1
M_3	1	0	0	0	0

FIGURE 3. Position matrix (indirect representation)

4.2. **Particles velocity, *pbest* and *nbest*.** Velocity of each particle is considered as a $m \times n$ matrix whose elements are in range $[-V_{\max},\ V_{\max}]$. In other words if V_k is the velocity matrix of kth particle, then:

$$V_k(i,j) \in [-V_{\max},\ V_{\max}] \quad (\forall i,j), \quad i \in \{1,2,...m\}, \quad j \in \{1,2,...,n\} \tag{8}$$

In order to represent *pbest* and *nbest*, direct and indirect encoding could be used the same as particles positions. The selection of representation method of *pbest* and *nbest* depends on particle position representation method; if direct encoding is used to represent particles, then the same encoding must be used to represent *pbest* and *nbest*, and in the same way if indirect encoding is used, the same encoding method should be applied to *pbest* and *nbest*. In indirect encoding *pbest* and *nbest* are $m \times n$ matrices and their elements are 0 or 1 as position matrices and in direct encoding *pbest* and *nbest* are $1 \times n$ vectors the same as position vectors. $pbest_k$ represents the best position that kth particle has visited since the first time step and $nbest_k$ represents the best position which kth particle and its neighbors have visited from the beginning of the algorithm. In each time step *pbest* and *nbest* should be updated; first fitness value of each particle (for exampleX_k) is estimated and in the case its value is greater than the fitness value of $pbest_k$ (*pbest* associated with X_k), $pbest_k$ is replaced with X_k. For updating *nbest* in each neighborhood, *pbest*s are used so that if in the neighborhood, fitness value of *pbest* is greater than *nbest*, then *nbest* is replaced with *pbest*. In the present paper, star topology is used to describe the neighborhood of particles (global best).

4.3. **Particle updating.** First we explain updating of the particles for the case of indirect encoding and then we will derive a method for updating particles in direct encoding. For the case of indirect encoding Equation (9) is used for updating the velocity matrix and then Equation (10) for position matrix of each particle.

$$V_k^{(t+1)}(i,j) = V_k^t(i,j) + c_1 r_1 (pbest_k^t(i,j) - X_k^t(i,j)) + c_2 r_2 (nbest_k^t(i,j) - X_k^t(i,j)) \tag{9}$$

$$X_k^{(t+1)}(i,j) = \begin{cases} 1 & \text{if } (V_k^{(t+1)}(i,j) = \max\{V_k^{(t+1)}(i,j)\}),\ \forall i \in \{1,2,...,m\} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In Equation (9) $V_k^t(i,j)$ is the element in ith row and jth column of the kth velocity matrix in tth time step of the algorithm and $X_k^t(i,j)$ denotes the element in ith row and

jth column of the kth position matrix in tth time step. Equation (10) means that in each column of position matrix, value 1 is assigned to the element whose corresponding element in velocity matrix has the maximum value in its corresponding column. If in a column of velocity matrix there is more than one element with maximum value, then one of these elements is selected randomly and 1 assigned to its corresponding element in the position matrix.

Example 1. Assume in a problem with 5 jobs and 3 machines $X_k = (1, 2, 2, 1, 3)$, $pbest_k = (2, 2, 1, 3, 2)$, $c_1 = 2$, $c_2 = 0$, and $r_1 = r_2 = 1$. Regarding the presented setting, only *pbest* is taken into account for velocity updating. Assume the velocity matrix, V_k is generated randomly as Figure 4.

	J_1	J_2	J_3	J_4	J_5
M_1	4	–2	4	4	–3
M_2	1	3	5	0	–2
M_3	2	–1	–3	–3	1

FIGURE 4. V_k, the velocity matrix in the above example

For velocity updating, first position vector (X_k) and *pbest* vector ($pbest_k$) must be converted to their equivalent position matrix as shown in Figures 5 and 6, and then using Equation (9) we can update velocity matrix as shown in Figure 7. Using updated velocity matrix and Equation (10) the position matrix can be estimated as shown in Figure 8. Also the obtained position matrix can be converted to $X_k = (2, 2, 1, 1, 2)$ in direct encoding.

	J_1	J_2	J_3	J_4	J_5
M_1	1	0	0	1	0
M_2	0	1	1	0	0
M_3	0	0	0	0	1

FIGURE 5. Equivalent position matrix with X_k

	J_1	J_2	J_3	J_4	J_5
M_1	0	0	1	0	0
M_2	1	1	0	0	1
M_3	0	0	0	1	0

FIGURE 6. Equivalent position matrix with $pbest_k$

	J_1	J_2	J_3	J_4	J_5
M_1	2	–2	6	2	–3
M_2	3	3	3	0	0
M_3	2	–1	–3	–1	–1

FIGURE 7. Updated velocity matrix

	J_1	J_2	J_3	J_4	J_5
M_1	0	0	1	1	0
M_2	1	1	0	0	1
M_3	0	0	0	0	0

FIGURE 8. Updated position matrix

As it can be seen in this example the particle has moved toward *pbets* and in case the updating process is repeated once more, the particle position will be the same as *pbest* position. Converting particles position from direct encoding to the indirect one or indirect encoding to the direct one is a time consuming operation. Using the above example we present a method for updating particles in direct encoding without converting particles into indirect encoding. As mentioned in Section 4.1 the value of $X_k(i, j)$ and $pbest_k(i, j)$ could be 0 or 1. Hence $c_1 r_1$ $(pbest_k(i, j) - X_k(i, j))$ in Equation (9) can be 0, $+(c_1 \times r_1)$, or $-(c_1 \times r_1)$. In the above example if $X_k(i, j) = pbest_k(i, j)$, then the corresponding element in velocity matrix, $V_k(i, j)$, does not change and if $X_k(i, j) \neq pbest_k(i, j)$, then two cases arise:

1) $pbest_k(i,j) = 1$ and $X_k(i,j) = 0$; in this case: $V_k(i,j) = V_k(i,j) + c_1 \times r_1$
2) $pbest_k(i,j) = 0$ and $X_k(i,j) = 1$; in this case: $V_k(i,j) = V_k(i,j) - c_1 \times r_1$

$pbest_k(i,j) = 1$ in direct encoding means $pbest_k(j) = i$ and $X_k(i,j) = 1$ means $X_k(j) = i$. According to the above points, if $pbest_k(j) = z$ and $X_k(j) = q$ and $z \neq q$ then $c_1 \times r_1$ is added to $V_k(z,j)$ and subtracted from $V_k(q,j)$. Regarding the above mentioned points and Equation (9) the updating algorithm for velocity matrix using direct encoding will be as follow:

```
for each particle k = 1, ..., P do
    for each job j = 1, ..., n do
        q = X_k(j);
        z = pbest_k(j);
        s = gbest_k(j);
        if q ≠ z then
            V_k(q,j) = V_k(q,j) − c_1 × r_1;
            V_k(z,j) = V_k(z,j) + c_1 × r_1;
        end
        if q ≠ s then
            V_k(q,j) = V_k(q,j) − c_2 × r_2;
            V_k(s,j) = V_k(s,j) + c_2 × r_2;
        end
    end
end
```

FIGURE 9. The algorithm for velocity updating in direct encoding

After updating the velocity matrix using the mentioned algorithm, the position vector of each particle is obtained as follows: For each element of position vector, a machine from its corresponding column in velocity matrix is chosen which has the maximum value in that column. Formally:

$$\begin{array}{ll} X_k(j) = \varphi \quad \text{if} \quad V_k(\varphi, j) = \max\{V_k(i,j)\} \\ \forall j \in (1,2,...,n) \qquad\qquad \forall i \in (1,2,...,m) \end{array} \tag{11}$$

If in a column of velocity matrix there is more than one element with maximum value, then one of machines which corresponds these elements is selected randomly. In our proposed method we use direct representation for particles because it is faster than the indirect method.

4.4. **Fitness evaluation.** In this paper makespan and flowtime are used to evaluate the performance of scheduler simultaneously. Because makespan and flowtime values are in incomparable ranges and the flowtime has a higher magnitude order over the makespan, the value of mean flowtime, *flowtime/m*, is used to evaluate flowtime where m is the number of available machines. The fitness value of each solution can be estimated using Equation (12).

$$fitness = (\lambda.makespan + (1-\lambda).mean\_flowtime)^{-1} \tag{12}$$

λ in Equation (12) is used to regulate the effectiveness of parameters used in this equation. $\lambda = 0$ means that only flowtime is considered in fitness evaluation and $\lambda = 1$ means that only makespan is considered in fitness evaluation. Also $\lambda \in (0, 1)$ means that both parameters are taken into account in fitness estimation. The greater λ, the more attention is paid by the scheduler in minimizing makespan and vise versa. The smaller makespan

and flowtime in Equation (12), the grater fitness value and hence a better solution it is regarded. Since makespan is the primary objective in grid systems, in this paper the value of $\lambda = 0.7$ has been fixed after a preliminary tuning process.

The pseudo code of the proposed PSO algorithm can be stated as Figure 10.

5. **Implementation and Experimental Results.** In order to evaluate the performance of the proposed method, the approach was compared with genetic algorithm (GA) [15], ant colony optimization (ACO) [36], fuzzy particle swarm algorithm (FPSO) [16] and continuous particle swarm optimization (CPSO) [25]. The first three are proposed for grid job scheduling and the fourth is used for task assignment problem in multiprocessor systems.

These methods are implemented using VC++ and run on a Pentium IV 3.2 GHz PC as well as ours. In order to optimize the performance of the proposed method and other methods, fine tuning has been performed and best values for their parameters are selected. For the proposed method the following ranges of parameter values were tested: $c1$ and $c2 = [1, 3]$, $P = [10, 100]$, $V_{\max} = [10, 100]$. Based on experimental results the proposed PSO algorithm performs best under the following settings: $c1 = c2 = 2.0$, $P = 50$, $V_{\max} = 40$.

5.1. **Benchmark problem.** We used the benchmark in [15] for simulating the heterogeneous computing environments. This model is based on expected time to compute (ETC) matrix for 512 jobs and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the three following metrics: job heterogeneity, machine heterogeneity, and consistency. In ETC matrix the amount of variance among the execution times of jobs for a given machine is defined as job heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given job across all the machines. Also an ETC matrix is said to be consistent whenever a machine M_i executes any job J_j faster than machine M_k; in this case, machine M_i executes all jobs faster than machine M_k. In contrast, inconsistent matrices characterize the situation where machine M_i may be faster than machine M_k for some jobs and slower for others. Partially-consistent matrices are inconsistent matrices that include a consistent sub-matrix of a predefined size [15]. Instances consist of 512 jobs and 16 machines and are labeled as u-x-yy-zz as follows:

- u means uniform distribution used in generating the matrices.
- x shows the type of inconsistency; c means consistent, i means inconsistent and p means partially-consistent.
- yy indicates the heterogeneity of the jobs; hi means high and lo means low.
- zz represents the heterogeneity of the machines; hi means high and lo means low.

For example "u-i-lo-hi" means an inconsistent environment with low heterogeneity in jobs and high heterogeneity in machines. This benchmark is used in some previous works (e.g. [12,14,15,29,35]) for simulating heterogeneous computing systems.

5.2. **Experimental results.** In our experiments, for generating initial population of the compared methods one solution is generated using min-min heuristic (that can achieve a very good reduction in makespan and flowtime [12,15]) and the others generated randomly from a uniform distribution. The statistical results of over 50 independent runs are compared in Table 1 for makespan. In this table the first column indicates the instance name, the second, third, fourth, fifth, sixth and seventh columns indicate the achieved value by min-min, GA [15], ACO [36], CPSO [25], FPSO [16] and proposed method (DPSO) respectively. Also Figure 11 shows the geometric mean of achieved makespan for 12 instances. As can be seen from Table 1 and Figure 11 our proposed method can

```
Create and initialize a D-dimensional swarm with P particles
//velocities are m × n matrices and X, pbest, gbest are 1 × n vectors (direct encoding)
repeat
    for each particle k = 1, ..., P do
        if f(X_k) > f(pbest_k) then // f() evaluates fitness function (Equation (12))
          pbest_k = X_k;
        end
        if f(pbest_k) > f(nbest_k) then
          nbest_k = pbest_k;
        end
    end
    for each particle k = 1, ..., P do
        for each job j = 1, ..., n do
          q = X_k(j);
          z = pbest_k(j);
          s = gbest_k(j);
          if  q ≠ z then
            V_k(q, j) = V_k(q, j) − c_1 × r_1;
            V_k(z, j) = V_k(z, j) + c_1 × r_1;
          end
          if  q ≠ s  then
            V_k(q, j) = V_k(q, j) − c_2 × r_2;
            V_k(s, j) = V_k(s, j) + c_2 × r_2;
          end
        end
        for each job j = 1, ..., n do
          if (∀i ∈ (1, 2, ..., m)) V_k(φ, j) = max{V_k(i, j)} then
            X_k(j) = φ;
          end
        end
    end
until stopping condition is true;
```

FIGURE 10. Pseudo code of the proposed method

achieve best results over makespan in most cases. FPSO and ACO can achieve admissible results in most cases too. Also we can see that GA is the best choice when we have a low heterogeneity in jobs and machines. It is evident that CPSO can not obtain a good reduction in makespan. This is because of the discrete nature of the benchmark which we used.

Table 2 depicts the statistical results of over 50 independent runs for flowtime. Also Figure 12 shows the geometric means of flowtime for 12 cases. We realize that reducing in makespan amount can lead to increasing in flowtime and since in most cases CPSO can not reduce the makespan effectively, it can achieve an admissible reduction in flowtime. Also we can see that our proposed method obtains maximum amounts of flowtime in most cases. In this paper we considered the makespan as primary objective (note that we set $\lambda = 0.7$ in fitness function) and as a result our proposed method tries to minimize makespan as primary objective which leads to increasing in flowtime amount.

Figure 13 shows the geometric mean of fitness values of compared methods over 12 mentioned cases. It is evident that our proposed method can obtain highest fitness values. Also ACO and FPSO are in the next ranks.

TABLE 1. Comparison of statistical results between our proposed method and others over makespan

Instance	min-min	GA	ACO	CPSO	FPSO	DPSO
u-c-hi-hi	8145395	7921686	7874958	8097772	7903366	**7741006**
u-c-hi-lo	164490	161923	161404	162324	161289	**161035**
u-c-lo-hi	279651	275260	274290	275565	272596	**268957**
u-c-lo-lo	5468	**5293**	5352	5361	5295	5332
u-i-hi-hi	3573987	3371842	**3235156**	3406340	3286295	3284287
u-i-hi-lo	82936	80620	80732	80936	80780	**79982**
u-i-lo-hi	113944	109832	109021	111944	110752	**107276**
u-i-lo-lo	2734	**2619**	2635	2703	2628	2652
u-p-hi-hi	4701249	4576995	4533476	4609927	4539559	**4479443**
u-p-hi-lo	106322	103961	104186	105221	**103349**	103711
u-p-lo-hi	157307	152046	153042	153913	153688	**149437**
u-p-lo-lo	3599	**3412**	3466	3505	3439	3416

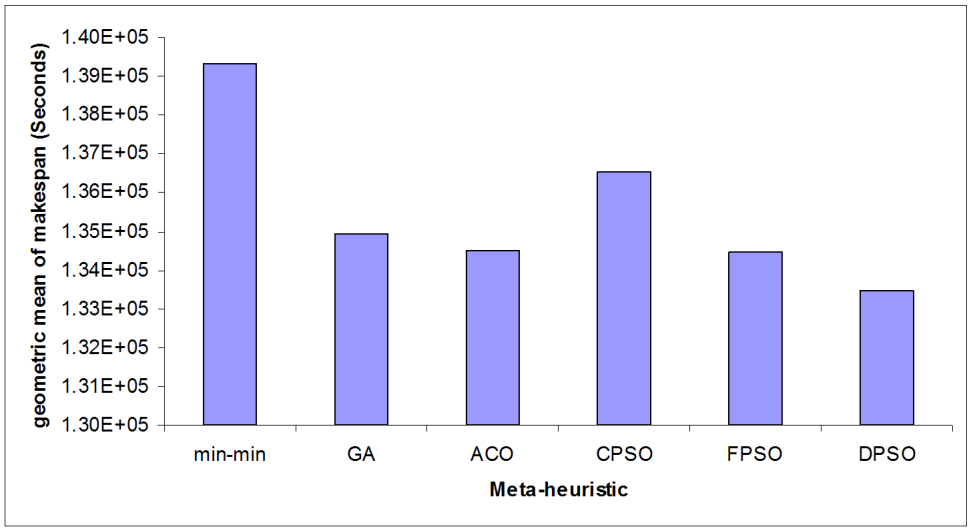

FIGURE 11. Geometric mean of makespan

TABLE 2. Comparison of statistical results between our proposed method and others over flowtime

Instance	min-min	GA	ACO	CPSO	FPSO	DPSO
u-c-hi-hi	115162284	115715373	116660362	**114960061**	116156174	115721406
u-c-hi-lo	2480404	2477784	2483428	**2468163**	2483921	2480406
u-c-lo-hi	3918515	3902067	3952086	**3881783**	3926216	3955765
u-c-lo-lo	80354	81210	80539	**79819**	81382	80787
u-i-hi-hi	**45696141**	46728285	46248373	46206878	47078048	47116485
u-i-hi-lo	**1214038**	1228876	1216711	1215087	1220823	1220894
u-i-lo-hi	1577886	1603161	1582819	**1577183**	1593862	1599217
u-i-lo-lo	39605	40295	39644	**39583**	40002	39904
u-p-hi-hi	**63516912**	64138008	64235803	63620547	64664919	64988191
u-p-hi-lo	**1565877**	1574007	1575167	1565925	1582217	1568909
u-p-lo-hi	**2118116**	2154052	2141610	2138413	2135065	2163470
u-p-lo-lo	51399	52129	51537	**51256**	52020	52074

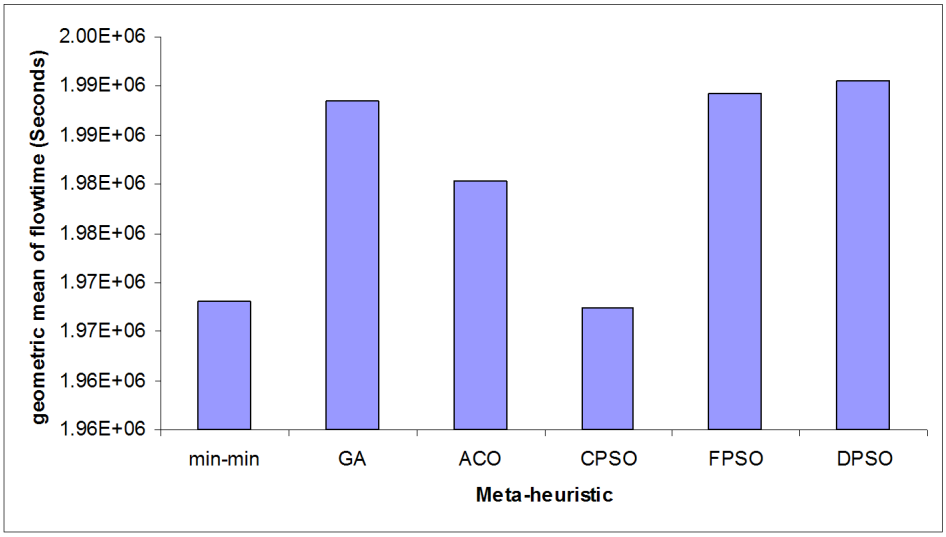

FIGURE 12. Geometric mean of flowtime

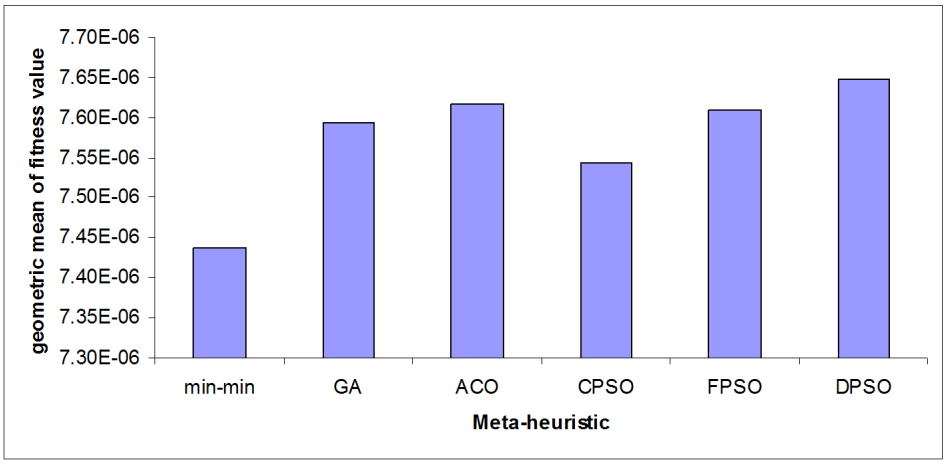

FIGURE 13. Geometric mean of obtained fitness values

Figure 14 illustrates the mean of standard deviation. CPSO and ACO have the lowest standard deviation while our proposed method has an admissible standard deviation (lower than 1.5%). Also Figure 15 shows a comparison of CPU times required to achieve results between compared methods. It is evident that the proposed method needs the lowest time for convergence in most cases, but by increasing the number of tasks and problem search space, the time for achieving results is increased in the proposed method rather than ACO and GA and in case of 1024 tasks, the ACO and GA schedulers need lower time for convergence.

6. **Conclusions.** Because of the heterogeneous and dynamic nature of grid environment we can not use the conventional scheduling methods in grid. This paper presents a version of Discrete Particle Swarm Optimization (DPSO) algorithm for grid job scheduling. Scheduler aims minimizing makespan and flowtime simultaneously. The performance of the proposed method was compared to those reported in recent work which used GA, ACO, fuzzy PSO and continuous PSO through carrying out exhaustive simulation tests and different settings. Experimental results show that our proposed method surpasses

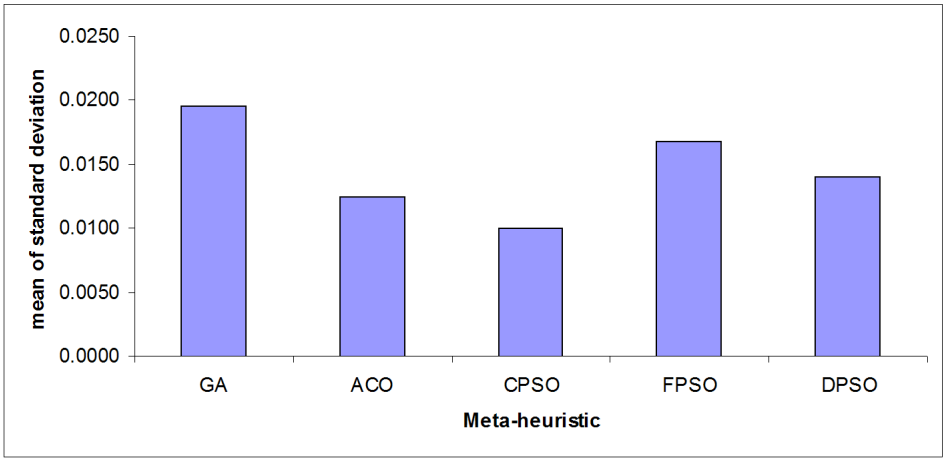

FIGURE 14. Mean of standard deviation for compared methods

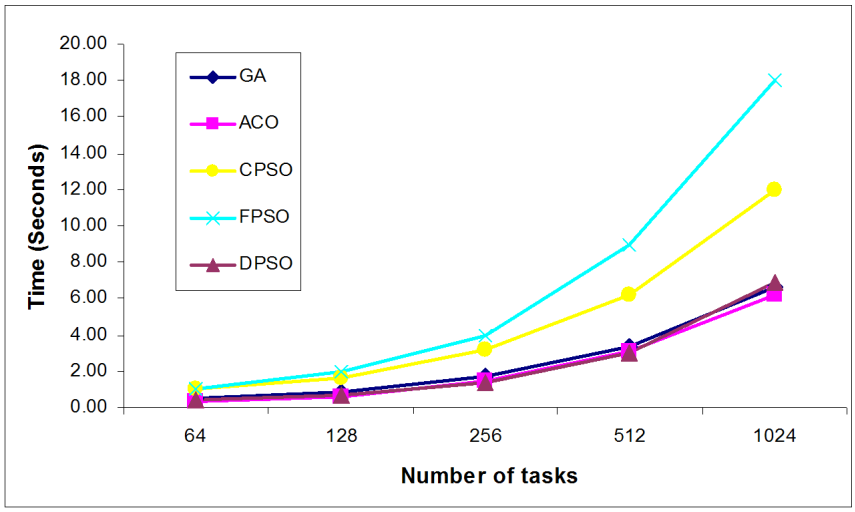

FIGURE 15. Comparison of CPU times required to achieve results

others in most cases. In future we aim at using our proposed method for grid job scheduling with a more quality of service constraints.

REFERENCES

[1] I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, *International Journal of High Performance Computing Applications*, vol.15, no.3, pp.200-222, 2001.

[2] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari and S. S. Yellampalli, Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment, *J. Parallel Distrib. Comp.*, vol.67, no.2, pp.154-169, 2007.

[3] J. Cao, *Agent-based Resource Management for Grid Computing*, Ph.D. Thesis, Department of Computer Science University of Warwick, London, 2001.

[4] R. Buyya, *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, Ph.D. Thesis, School of Computer Science and Software Engineering Monash University, Melbourne, 2002.

[5] Y. He, H. Liu, W. Wen and H. Jin, A preference method with fuzzy logic in service scheduling of grid computing, *Lecture Notes in Computer Science*, vol.3613, pp.865-871, 2005.

[6] C. Wang, C. Jiang and X. Liu, Fuzzy logic-based secure and fault tolerant job scheduling in grid, *Tsinghua Science and Technology*, vol.12, supplement 1, pp.45-50, 2007.
[7] K. P. Kumar, A. Agarwal and R. Krishnan, Fuzzy based resource management framework for high throughput computing, *Proc. of the 4th IEEE International Symposium on Cluster Computing and the Grid*, Chicago, IL, vol.3, pp.555-562, 2004.
[8] E. G. Coffman, *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons Inc, New York, 1976.
[9] R. Zhang and C. Wu, Bottleneck machine identification based on optimization for the job shop scheduling problem, *ICIC Express Letters*, vol.2, no.2, pp.175-180, 2008.
[10] A. Yarkhan and J. Dongarra, Experiments with scheduling using simulated annealing in a grid environment, *Proc. of the 3rd International Workshop on Grid Computing, LNCS 2536*, pp.232-242, 2002.
[11] W. Pang, K. Wang, C. Zhou and L. Dong, Fuzzy discrete particle swarm optimization for solving traveling salesman problem, *Proc. of the 4th International Conference on Computer and Information Technology*, pp.796-800, 2004.
[12] H. Izakian, A. Abraham and V. Snášel, Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments, *Neural Network World*, vol.19, no.6, pp.695-710, 2009.
[13] J. Page and J. Naughton, Framework for task scheduling in heterogeneous distributed computing using genetic algorithms, *Artif. Intell. Rev*, vol.24, no.3-4, pp.415-429, 2005.
[14] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, in *Advances in Computers Volume 63, Parallel, Distributed, and Pervasive Computing*, A. R. Hurson (ed.), Elsevier, 2005.
[15] T. D. Braun, H. J. Siege, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, *J. Parall. Distrib. Comp.*, vol.61, no.6, pp.810-837, 2001.
[16] H. Liu, A. Abraham and A. E. Hassanien, Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm, *Future Generation Computer Systems*, 2009, in press.
[17] J. Kennedy and R. C. Eberhart, Particle swarm optimization, *Proc. of the IEEE International Conference on Neural Networks*, pp.1942-1948, 1995.
[18] *SETI*, http://setiathome.ssl.berkeley.edu.
[19] J. Kennedy, Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance, *Proc. of the Congress on Evolutionary Computation*, Washington DC, USA, pp.1931-1938, 1999.
[20] R. C. Eberhart and Y. Shi, Evolving artificial neural networks, *Proc. of the International Conference on Neural Networks and Brain*, Beijing, China, pp.PL5-PL13, 1998.
[21] R. C. Eberhart and J. Kennedy, A new optimizer using particle swarm theory, *Proc. of the 6th International Symposium on Micromachine and Human Science*, pp.39-43, 1995.
[22] J. Kennedy and R. C. Eberhart, A discrete binary version of the particle swarm algorithm, *IEEE International Conference on Systems, Man, and Cybernetics*, Orlando, FL, vol.5, pp.4104-4108, 1997.
[23] S. Kirkpatrick, C. Gelatt Jr. and M. Vecchi, Optimization by simulated annealing, *Science*, vol.220, no.4598, pp.671-680, 1983.
[24] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, Reading, 1997.
[25] A. Salman, I. Ahmad and S. Al-Madani, Particle swarm optimization for task assignment problem, *Microprocessors and Microsystems*, vol.26, no.8, pp.363-371, 2002.
[26] M. Dorigo, *Optimization, Learning, and Natural Algorithms*, Ph.D. Thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Italy, 1992.
[27] M. Litzkow, M. Livny and M. W. Mutka, Condor–a hunter of idle workstations, *Proc. of the 8th Int'l Conf. Distributed Computing Systems*, San Jose, CA, pp.104-111, 1988.
[28] A. Abraham, R. Buyya and B. Nath, Nature's heuristics for scheduling jobs on computational grids, *Proc. of the 8th IEEE International Conference on Advanced Computing and Communications*, India, pp.45-52, 2000.
[29] J. Carretero, F. Xhafa and A. Abraham, Genetic algorithm based schedulers for grid computing, *International Journal of Innovative Computing, Information and Control*, vol.3, no.5, pp.1053-1071, 2007.

[30] J. Li, D. Wan, Z. Chi and X. Hu, An efficient fine-grained parallel particle swarm optimization method based on GPU-acceleration, *International Journal of Innovative Computing, Information and Control*, vol.3, no.6(B), pp.1707-1714, 2007.

[31] X. Cai, Z. Cui, J. Zeng and Y. Tan, Particle swarm optimization with self-adjusting cognitive selection strategy, *International Journal of Innovative Computing, Information and Control*, vol.4, no.4, pp.943-952, 2008.

[32] R. C. Eberhart, P. K. Simpson and R. W. Dobbins, *Computational Intelligence PC Tools*, Academic Press Professional, San Diego, 1996.

[33] J. Salerno, Using the particle swarm optimization technique to train a recurrent neural model, *Proc. of the 9th IEEE International Conference on Tools with Artificial Intelligence*, Newport Beach, CA, pp.45-49, 1997.

[34] J. Kennedy, The particle swarm: Social adaptation of knowledge, *Proc. of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, pp.303-308, 1997.

[35] F. Xhafa, B. Duran, A. Abraham and K. Dahal, Tuning struggle strategy in genetic algorithms for scheduling in computational grids, *Neural Network World*, vol.18, no.3, pp.209-225, 2008.

[36] G. Ritchie and J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, *Proc. of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*, Cork, Ireland, pp.1-7, 2004.

[37] J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery and J. Zayer, A world wide number field sieve factoring record: On to 512 bits, *Advances in Cryptology—Proc. Int. Conf. Theory and Applications of Cryptology and Information Security, LNCS 1163*, pp.382-394, 1996.

[38] *Prime*, http://www.mersenne.org.

[39] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su and D. Zagorodnov, Adaptive computing on the grid using AppLeS, *IEEE Trans. Parallel and Distributed Systems*, vol.14, no.4, pp.369-382, 2003.

[40] Z. Cui, J. Zeng and G. Sun, Levy velocity threshold particle swarm optimization, *ICIC Express Letters*, vol.2, no.1, pp.23-28, 2008.