A Novel Particle Swarm Optimization Approach for Grid Job Scheduling

Hesam Izakian[1], Behrouz Tork Ladani[2], Kamran Zamanifar[2], Ajith Abraham[3]

[1]Islamic Azad University, Ramsar branch, Ramsar, Iran
izakian@gmail.com
[2]Department of Computer Engineering, University of Isfahan, Isfahan, Iran
{Ladani, zamanifar}@eng.ui.ac.ir
[3]Norwegian Center of Excellence, Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, Norway
ajith.abraham@ieee.org

Abstract

This paper represents a Particle Swarm Optimization (PSO) algorithm, 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 we used a PSO approach for grid job scheduling. The scheduler aims at minimizing makespan and flowtime simultaneously. Experimental studies show that the proposed novel approach is more efficient than the PSO approach reported in the literature.

1. Introduction

Computational Grid [1] is composed of a set of virtual organizations (VOs). Any 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 and services are added to or removed from grid and as a result, grid environment is highly dynamic.

Service is an important concept in many distributed computations and communications. Service is used to depict the details of a resource within the grid [2]. Grid services and resources are registered within one or more Grid Information Servers (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. Until now a lot of works has been done in order to schedule jobs in a computational grid. Yet according to the new nature of the subject further research is required. Cao [3] used agents to schedule grid. 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.

As mentioned in [8] scheduling is NP-complete. Meta-heuristic methods have been used to solve well known NP-complete problems. In [10] Yarkhanan and Dongarra used simulated annealing for grid job scheduling. GAs for grid job scheduling is addressed in several works [12], [13], [14] and [16]. Abraham et al. [15] used fuzzy PSO for grid job scheduling.

Different criteria can be used for evaluating the efficacy of scheduling algorithms and 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 [15]. The method proposed in [15] 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 method presented in [15] in order to evaluate its efficacy. The experimental results show the presented method is more efficient 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 the PSO paradigm is briefly discussed and Section 4 describes the proposed method for grid job scheduling, and Section 5 reports the experimental results. Finally Section 6 concludes this work.

2. Problem Formulation

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 nodes $\{N_1, N_2, ..., N_m\}$ and k services, $\{S_1, S_2, ..., S_k\}$ are within the grid. Each node has one or more services and each job requires one service. If a job requires more than one independent service, then we can consider it as a set of sub-jobs each requiring a service.

In this paper, scheduling is done at node level and it is assumed that each node uses First-Come, First-Served (FCFS) method for performing the received jobs. We assume that each node in the grid can estimate how much time is needed to perform each service it includes. In addition each node includes a time as previous workload which is the time required for performing the jobs given to it in the previous steps. We used the ETC model to estimate the required time for executing a job in a node. 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.

Table 1 shows a simple example of a set of received jobs by GRB in a specific time interval and status of available nodes and services in the grid. In this Table GRB received 5 jobs in a time interval and the status of available nodes and resources in the grid is as follows:

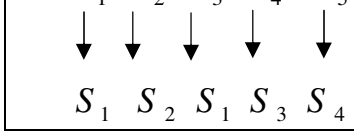
Table 1
A simple example of a set of jobs and grid status

$Jobs: \{J_1, J_2, J_3, J_4, J_5\}$

$Services: \{S_1, S_2, S_3, S_4\}$

$Nodes: \{N_1, N_2, N_3\}$

Jobs Requirements:



Nodes status:

	Previous workload	S_1	S_2	S_3	S_4
N_1	15	18	5	75	∞
N_2	60	∞	17	∞	15
N_3	36	12	∞	98	∞

There are 4 services and 3 nodes; N_1 includes 15 time units of previous workload which means that it requires 15 time units to complete the tasks already submitted to it. This node requires 18 time units to perform S_1, 5 time units to perform S_2 and 75 time units to perform S_3. Since this node does not include S_4, it is not able to perform it. Therefore the required time to perform S_4 by this node is considered as ∞. In this Table job J_1 requires service S_1, J_2 requires S_2, J_3 requires S_1 , J_4 requires S_3, and J_5 requires S_4. Scheduling algorithm should be designed in a way that each job is allocated to a node which includes the services required by that job.

Assume that $C_{i,j}$ $(i \in \{1,2,...,m\}, j \in \{1,2,...,n\})$ is the completion time for performing *j*th job in *i*th node and W_i $(i \in \{1,2,...,m\})$ is the previous workload of N_i, then Eq. (1) shows the time required for N_i to complete the jobs included in it. According to the aforementioned definition, makespan and flowtime can be estimated using Equations. (2) and (3) respectively.

$$\sum C_i + W_i \tag{1}$$

$$makespan = \max\{\sum C_i + W_i\}, \quad i \in \{1,2,...,m\} \tag{2}$$

$$flowtime = \sum_{i=1}^{m} C_i \tag{3}$$

As mentioned in the previous section the goal of the scheduler 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 [9] in 1995. In contrast to 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 [5].

In 1997 the binary version of this algorithm was presented by Kennedy and Eberhart [6] 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. When all of the population size of the swarm is considered as the neighbor of a particle, *nbest* is called global best (star neighborhood topology) and if the smaller neighborhoods are defined for each particle (e.g. ring neighborhood topology), then *nbest* is called local. Equations 4 and 5 are used to update the velocity and position vectors of the particles respectively.

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

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

where,

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

In (4) $X_i^t(j)$ is *j*th element of *i*th particle in *t*th step of the algorithm and $V_i^t(j)$ is the *j*th element of the velocity vector of the *i*th particle in *t*th 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. w which is called inertia weight was introduced by Shi and Eberhart [7] as a mechanism to control the exploration and exploitation abilities of the swarm. Usually *w* starts with large values (e.g. 0.9) which decreases over time to smaller values so that in the last iteration it ends to a small value (e.g. 0.1). r_{ij} in Eq. (5) is a random number in range $[0, 1]$ and Eq. (6) shows sigmoid function.

4. Proposed PSO Algorithm for Grid Job Scheduling

In this section we propose a version of discrete particle swarm optimization for grid job scheduling. Particle needs to be designed to present a sequence of jobs in available grid nodes. Also the velocity has to be redefined. Details are given what follows.

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. In our method solutions are encoded in a $m \times n$ matrix, called position matrix, in which *m* is the number of available nodes 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 *k*th particles, then:

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

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 node. In each column it is determined that a job should be performed by which node. Assume that X_k shows the position matrix of *k*th particle. If $X_k(i,j)=1$ then the *j*th job will be performed by *i*th node. Figure 1 shows a position matrix in the example mentioned in Table 1. This position matrix shows that J_2 and J_4 will be performed in N_1; J_3 and J_5 will be performed in N_2 and J_1 will be performed in N_3.

	J_1	J_2	J_3	J_4	J_5
N_1	0	1	0	1	0
N_2	0	0	1	0	1
N_3	1	0	0	0	0

Fig. 1. Position matrix

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 *k*th particle, then:

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

Also *Pbest* and *nbest* are $m \times n$ matrices and their elements are 0 or 1 as position matrices. $pbest_k$ represents the best position that *k*th particle has visited since the first time step and $nbest_k$ represents the best position that *k*th particle and its neighbors have visited from the beginning of the algorithm. In this paper we used star neighborhood topology for *nbest* . In each time step *pbest* and *nbest* should be updated; first fitness value of each particle (for example X_k) is estimated and in 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, *pbests* are used so that if in neighborhood fitness value of *pbest* is greater than *nbest* , then *nbest* is replaced with *pbest* .

4.3. Particle updating

Equation (9) is used for updating the velocity matrix and then (10) for position matrix of each particle.

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

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

In (9) $V_k^t(i,j)$ is the element in *i*th row and *j*th column of the *k*th velocity matrix in *t*th time step of the algorithm and $X_k^t(i,j)$ denotes the element in *i*th row and *j*th column of the *k*th position matrix in *t*th time step. Eq. (10) means that in each column of position matrix value 1 is assigned to the element whose corresponding element in velocity matrix has the max value in its corresponding column. If in a column of velocity matrix there is more than one element with max value, then one of these elements is selected randomly and 1 assigned to its corresponding element in the position matrix.

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 nodes. The Fitness value of each solution can be estimated using (11).

$$fitness = (\lambda . makespan + (1-\lambda). mean\_flowtime)^{-1}\,, \quad \lambda \in [0,1] \quad (11)$$

λ in (11) is used to regulate the effectiveness of parameters used in this equation. The greater λ, more attention is paid by the scheduler in minimizing makespan and vise versa. The smaller makespan and flowtime in (11), the grater fitness value, and hence a better solution it is regarded.

4.5. Proposed PSO algorithm

The pseudo code of the proposed PSO algorithm is stated as follows:

```
Create and initialize a m×n -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 matrix using Eq. (9)
                update the position matrix using Eq. (10)
        end
until stopping condition is true;
```

Fig. 2. Pseudo code of the proposed method

5. Implementation and Experimental Results

In this Section, the proposed algorithm is compared to the methods presented in [15]. Both approaches were implemented using VC++ and run on a Pentium IV 3.2 GHz PC. In the preliminary experiment the following ranges of parameter values were tested: λ = [0, 1], c_1 and c_2 = [1, 3], $w = [1.4 \rightarrow 0.01]$, $P = [10, 30]$, $V_{\max} = [5, 50]$, and maximum iterations = $[20 \times m, 200 \times m]$ in which m is the number of nodes. Based on experimental results the proposed PSO algorithm performs best under the following settings: $\lambda = 0.35$, $c_1 = c_2 = 1.5$, $w = 0.9 \rightarrow 0.1$, $P = 28$, $V_{\max} = 30$, and maximum iteration $= 100 \times m$.

5.1. Comparison of results with the method proposed in [15]

Abraham et al. [15] used Fuzzy discrete particle swarm optimization [11] for grid job scheduling. In their method, the position of each particle is presented as $m \times n$ matrices in which m is the number of available nodes and n is the number of received jobs. Each matrix represents a potential solution whose elements are in [0, 1] intervals in which the total sum of the elements of each column is equal to 1. The value of s_{ij}, the element in *i*th row and *j*th column of the position matrix, means the degree of membership that the grid node N_j would process the job J_i in the feasible schedule solution [15]. In the first time step of the algorithm one position matrix is generated using LJFR-SJFR heuristic [16] that minimizes the makespan and the flowtime simultaneously and others are generated randomly and then in each time step these matrices are updated using velocity matrix whose elements are real numbers in range $[-V_{\max}, V_{\max}]$. After updating each position matrix, it is normalized in a way that each element is in range [0, 1] and the sum of values of each column equals 1 and then using these obtained matrices schedules are generated.

In this paper, for comparison and evaluation of the scheduler, makespan and mean flowtime are used simultaneously. A random number in the range $[0, 500]$, sampled from uniform distribution, is assigned to the previous workload of each node in our tests. One or more services (at most *k* services) of $\{S_1, S_2, ..., S_k\}$ are randomly selected for each node. The time for executing services is randomly selected in range $[1, 100]$ if the node has these services; otherwise it is selected as ∞. For each job one service among k services is selected randomly as the required service of that job. To improve the efficiency of our proposed method and the method presented in [15] we generate only feasible solutions in initial step as well as each iteration/generation. In other words each job is allocated to the node which has the service required by that job. If in grid there is a job that its corresponding service does not exist in any of the nodes, then its allocated node is considered as -1 and this means that this job is not performable in the grid at that specific time. In this case ∞, the required time for performing the job, is not taken into account in fitness estimation so that the efficiency of the method does not fade. Nine grid status of different sizes with number of jobs, *n*=50, 100, 300 number of nodes, *m*=10, 20, 30 and number of services, *k*=40, 80, 160 are generated. The statistical results of over 50 independent runs are illustrated in Table 2.

Table 2
Comparison of statistical results between our proposed method and FDPSO proposed in [15]

Case Study	Grid status: Number of (Jobs, Nodes, Services)	Number of iterations: $(100\times m)$	LJFR-SJFR heuristic		FPSO [15]		Proposed DPSO	
			makespan	flowtime	makespan	flowtime	makespan	flowtime
I	(50,10,40)	1000	607.9	1337.4	530.5	1252.2	**500.6**	**1186.8**
II	(100,10,80)	1000	750.3	2440.1	658.2	2309.1	**581.4**	**2139.5**
III	(300,10,160)	1000	1989.0	6956.8	1359.7	6769.3	**1226.7**	**6483.9**
IV	(50,20,40)	2000	482.7	1230.4	470.9	1057.0	**462.8**	**891.3**
V	(100,20,80)	2000	550.2	1881.6	511.3	1443.2	**497.4**	**1278.5**
VI	(300,20,160)	2000	886.9	4863.5	667.1	4215.7	**535.8**	**3830.9**
VII	(50,30,40)	3000	467.3	1177.1	468.6	821.5	**459.0**	**691.3**
VIII	(100,30,80)	3000	487.7	1603.4	468.9	1124.8	**443.5**	**983.1**
IX	(300,30,160)	3000	554.6	3691.2	533.5	3324.3	**490.2**	**2912.4**

As evident, the proposed method performs better than the Fuzzy PSO proposed in [15]. Figures 3 and 4 show a comparison of CPU time required to achieve results and the fitness values of each method for different case studies as shown in Table 2.

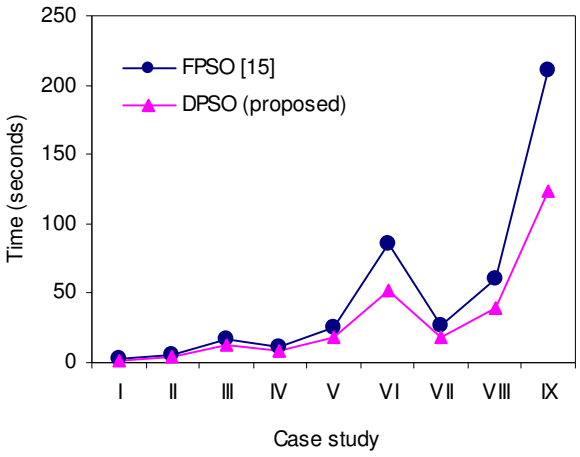


Fig. 3. Comparison of convergence time between our proposed method and FPSO [15] .

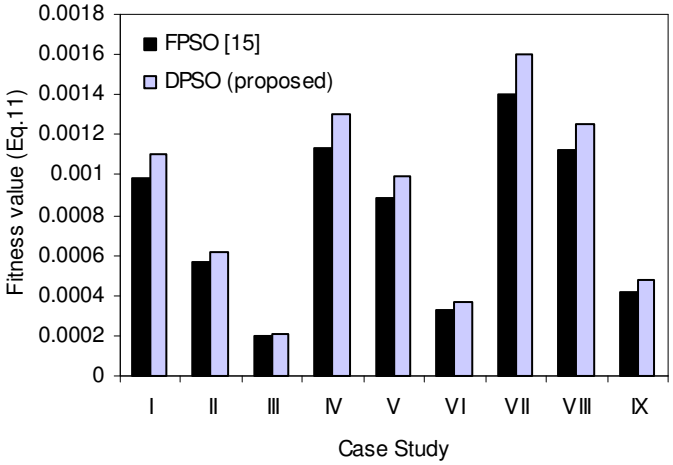


Fig. 4. Comparison of fitness values between our proposed method and FPSO [15].

6. Conclusions

This paper presented a version of Discrete Particle Swarm Optimization (DPSO) algorithm for grid job scheduling. Scheduler aims at generating feasible solutions while minimizing makespan and flowtime simultaneously. The performance of the proposed method was compared with the fuzzy PSO through carrying out exhaustive simulation tests and different settings. Experimental results show that the proposed method outperforms fuzzy PSO. In the future, we plan to use the proposed method for grid job scheduling with more quality of service constraints.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of High Performance Computing Applications 15 (2001) 200-222.

[2] J. Cao, D.J. Kerbyson, G.R. Nudd, Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing, in: Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (2001) 311-318.

[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] A. Salman, I. Ahmad, S. Al-Madani, Particle swarm optimization for task assignment problem, Microprocessors and Microsystems 26 (2002) 363–371.

[6] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, IEEE international conference on Systems, Man, and Cybernetics (1997) 4104 – 4108.

[7] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: proceedings of the IEEE Congress on Evolutionary Computation (1998) 69-73.

[8] E.G. Coffman Jr. (Ed.), Computer and Job-Shop Scheduling Theory, Wiley, New York, NY, 1976.

[9] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks (1995) 1942–1948.

[10] A. Yarkhan, J. Dongarra, Experiments with scheduling using simulated annealing in a grid environment, in: 3rd International Workshop on Grid Computing (2002) 232–242.

[11] W. Pang, K. Wang, C. Zhou, L. Dong,Fuzzy Discrete Particle Swarm Optimization for Solving Traveling Salesman Problem, In: Proceedings of the Fourth International Conference on Computer and Information Technology, IEEE CS Press (2004) 796–800.

[12] V. Di Martino, M. Mililotti, Sub optimal scheduling in a grid using genetic algorithms, Parallel Computing 30 (2004) 553–565.

[13] D. Liu, Y. Cao, CGA: Chaotic Genetic Algorithm for Fuzzy Job Scheduling in Grid Environment, Springer-Verlag Berlin Heidelberg (2007) 133–143.

[14] Y. Gao, H. Rong, J.Z. Huangc, Adaptive grid job scheduling with genetic algorithms, Future Generation Computer Systems 21 (2005) 151–161.

[15] A. Abraham, H. Liu, W. Zhang, T.G. Chang, Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm, Springer-Verlag Berlin Heidelberg (2006) 500-507.

[16] A. Abraham, R. Buyya, B. Nath, Nature's heuristics for scheduling jobs on computational grids, In: 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), India, ISBN 0070435480, Tata McGraw-Hill Publishing Co. Ltd, New Delhi, pp. 45-52, 2000.