A Hybrid Metaheuristic Algorithm for Job Scheduling on Computational Grids

Zahra Pooranian
Department of Computer Engineering, Dezful Branch, Islamic Azad University, Dezful, Iran
E-mail: Zahra.Pooranian@gmail.com

Mohammad Shojafar
Department of Information Engineering, Electronics and Telecommunication (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184, Rome, Italy
E-mail: Shojafar@diet.uniroma1.it

Reza Tavoli
Department of Mathematics, Islamic Azad University, Chalous Branch (IAUC)17Shahrivar Ave., P.O. Box 46615-397, Chalous, Iran
E-mail: r.tavoli@gmail.com

Mukesh Singhal
Computer Science & Engineering, University of California, Merced, CA S&E 296,USA
E-mail: msinghal@ucmerced.edu

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, P.O. Box 2259, Auburn, WA 98071-2259, USA
E-mail: Ajith.Abraham@ieee.org

Keywords: **g**rid computing, genetic algorithm, gravitational emulation local search (GELS), independent task scheduling.

Received: April 25, 2013

The dynamic nature of grid resources and the demands of users produce complexity in the grid scheduling problem that cannot be addressed by deterministic algorithms with polynomial complexity. One of the best methods for grid scheduling is the genetic algorithm (GA); the simple and parallel features of this algorithm make it applicable to several optimization problems. A GA searches the problem space globally and is unable to search locally. Therefore, scholars have investigated combining GAs with other meta-heuristic methods to resolve the local search problem. This is the focus of the present contribution, where we have developed a new hybrid scheduling algorithm GGA that combines GA and the gravitational emulation local search (GELS) algorithm. The noteworthy feature of the proposed optimal scheduler is that it decreases runtime and the number of submitted tasks whose deadlines are missed. A comparison of the performance of our proposed joint optimal scheduler to similar methods shows that it produces more optimal computation time.

Povzetek: Predlagana je metoda genetskih algoritmov za razvrščanje poslov v grid sistemih.

1 Introduction

Grid computing has emerged as a new approach for solving large-scale problems in scientific, engineering, and commercial fields [1].

A deciding factor in grid computing design is the purpose for which it will be used. Design goals can be divided into three major groups: increasing the efficiency of an application, improving data access, and increasing and improving services. Grid systems can be classified according to these objectives as, respectively, grid computing systems, data grids, and service grids. Further, grid-computing systems can be classified into two main categories: distributed supercomputing and high-throughput grids [2].

Data grids provide a platform for assembling new databases from distributed data sources such as digital libraries or providers' data warehouses. Although grid computing also needs to provide data services, the major difference between a data grid and grid computing is that the former provides a special platform to manage data storage and access for applications, while in grid

computing, the applications themselves must implement the storage management schema. An example use for data grids is data mining that gathers information from various sources. Two organizations that are working on developing large-scale data collections are the European data grid and Globus [2].

Systems in a service grid provide services that cannot be provided with a single machine [3]. Most research in grid computing falls under one of these classifications (data, computing, and service grids).

Among existing uses of grids, grid computing is the most prevalent. By utilizing the processing power of CPUs during their idle periods, grid computing can be several times faster than what a single computer can achieve today. Therefore, acceptable task scheduling for resources plays a crucial role in grids, especially scheduling computing resources for tasks. The main goal of most schedulers is to find a balance between execution cost and the runtime for tasks. This means that given a deadline for completing execution, the running costs are kept low, or given a fixed cost for execution; the necessary time to perform the tasks will be minimized.

Generally, there are three methods for scheduling:

1. Manual scheduling. The user divides the tasks between different resources.

2. Application-mode scheduling. Applications perform the scheduling, with each application defining the resources, such as MPI programs, required for its execution. A list of machines that have MPI programs is given to the user at runtime.

3.Scheduling that is independent of applications, such as scheduling by a grid broker. This method is much more appropriate for grid scheduling. For task processing and task analysis, applications deliver their requirements to the broker, based on the quality of service required for their tasks.

We should note that the resources for grid task scheduling are distributed in various locations. One or more resources are selected for running a task, which is then sent to those resources. The grid scheduler has no ownership or control over resources. Rather, tasks are delivered to local resource managers (LRMs) for execution. After that, the LRMs control the running status and execution of the tasks they have received.

The first phase of grid task scheduling is resource discovery, which generates a list of potential resources. The second phase includes gathering information about these resources and choosing the best set of resources matching the application's requirements. In the third phase, the task is executed, which involves file staging and cleanup [4].

Grid systems consist of heterogeneous resources, managerial systems, policies, and applications with different requirements. Since these resources are heterogeneous and distributed and are used in common, grid efficiency is highly dependent on an effective and efficient design for its scheduler. Grid scheduling is considered to be an NP-hard problem. Deterministic algorithms do not have the necessary efficiency for solving this problem. Therefore, much research has been directed toward heuristic methods. Most of these methods attempt to minimize makespan.

Many heuristic algorithms have recently been suggested for task scheduling in grid computing, including hierarchical stochastic Petri net schedulers (HSPNs) [5-8], genetic algorithms (GAs) [9], the group leaders' optimization algorithm (GLOA) [10], simulated annealing (SA) [11], the queen bee method [12], and the tabu search (TS) [13] and others [29-36]. Among these, GAs provide the best heuristic method because they are inherently parallel and can search several aspects of a problem space simultaneously. Since the convergence of a GA is slow for global optimization and has been proved to be unstable in different implementations, the efficiency of GAs can be improved by combining them with other algorithms such as GPSO [14] that it combines GELS method with PSO.

This research combines a GA and the gravitational emulation local search (GELS) algorithm. GA's are weak for local searches and strong for global searches. Conversely, GELS is a local search algorithm that imitates gravitational attraction and is therefore strong for local searches and weak for global searches. Combining the benefits of these two algorithms can solve the grid-scheduling problem. This paper presents a static scheduling algorithm for scheduling independent tasks in a grid system. "Static scheduling" means that all necessary data about tasks, resources, and the number of resources should be specified before execution. The advantage of static scheduling is that no overhead is exerted on the system. In addition to decreasing makespan, our proposed algorithm considers quality of service (QOS) to minimize the number of tasks that miss their deadlines.

The remainder of this paper is organized as follows. Section 2 briefly describes previous related work and the intelligent GA and the GELS algorithm, respectively. Section 3 describes the task-scheduling problem. Section 4 presents our proposed algorithm in detail. Section 5 compares our proposed algorithm with several similar algorithms, and Section 6 presents our conclusions and future research directions.

2 Related Work

In the following Section, we provide an overview of Genetic algorithm and GELS algorithm and explain various methods, which describe different hybrid, and joint method that applied for scheduling in grid computing.

2.1 Genetic algorithms

GAs were first proposed in 1975 by John Holland et al. [15] at Michigan University. In optimization methods, a GA or optimization inspired by nature is considered to be the most natural evaluation method. A GA selects the most suitable strings from organized stochastic information that is searched and gathered by humans. In each generation, a new set of strings is produced based on artificial strings with the help of the most suitable bits

and elements among the old elements. The new set is tested stochastically, and its strength or fitness level is evaluated. The general form of a GA is as follows:

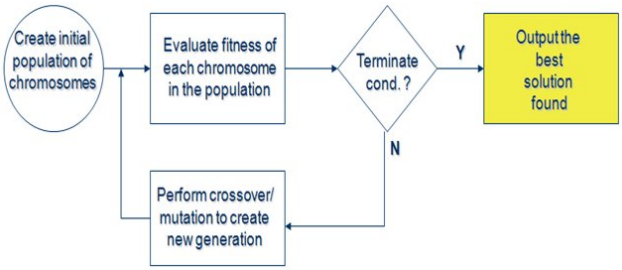

Figure 1: Genetic algorithm.

2.2 Gravitational Emulation Local Search

In 1995, Voudouris and his colleagues [16] proposed the Guided Local Search (GLS) algorithm for searching in a search space with an NP-hard solution. In 2004, Vebster [17] presented GLS as a strong algorithm, the GELS algorithm. GELS mimics gravitational attraction for searching within a search space. Each response has different neighbors that can be grouped based on problem-dependent criteria. The neighbors obtained in each neighbor group are called a *dimension*. A primary velocity is defined for each dimension, and a dimension with a greater primary velocity is more responsive for the problem. The GELS algorithm accounts for gravitational force in the responses in a search space through two methods. In the first method, a response is selected from the local neighbor space of the current response and the gravitational force between these two responses is calculated. In the second method, the gravitational force is calculated using all of the neighbor responses in a neighbor space of the current response rather than being limited to one response. GELS also implements movement into the search space with two methods. The first method allows movement from the current response toward the response to the current response in local neighbor spaces. The second method allows movement toward responses outside of the current response local neighbor spaces in addition to the neighbor responses of the current response. Each of these movement methods can be used in combination with each gravitation force calculation method, so that there are four models for the GELS algorithm.

In 2007, Blachandar [18] used the GELS algorithm to solve the Travelling Salesman Problem and compared it with other algorithms such as hill climbing and SA. The results showed that whenever the size of a problem is small, all algorithms perform roughly the same, but whenever the size of the problem is large, the GELS algorithm obtains better results than the other algorithms.

The algorithm begins with a primary response and a primary velocity vector that consists of a primary velocity specified by the user or randomly generated. After the primary velocity vector has been examined, the responsive dimension with the greatest primary velocity among the neighbor dimensions is selected for movement (select and obtain neighbor response).

The algorithm uses a pointer object that can move within the search space. This object always refers to the response with the most weight. In the first iteration of the algorithm using the first method, a dimension is selected for obtaining a neighbor response from the current response and a candidate response is selected from the local neighbor space of the current response in terms of this dimension. The gravitational force between the current and candidate responses is calculated and then added to the primary velocity of the dimension from which the candidate response was obtained. This is called the *updated primary velocity*. In the next iteration, the primary velocity vector is examined and a new movement direction is selected for continuing the response search. Each iteration of the algorithm using the second method is generally similar to the first method except that instead of calculating gravitational force and updating the primary velocity vector for just one candidate response in the current dimension, gravitational force is calculated and the primary velocity updated for each candidate response in the current dimension. In this algorithm, the gravitational force between two entities is calculated using Equation (1):

$$f = \frac{G(CU - CA)}{R^2} \tag{1}$$

where CA and CU are the candidate response and current response, respectively; G is the constant 6.672; and R is the neighbor radius of two parameters in the search space. R may be constant or can change intelligently in each iteration. The algorithm terminates when one of the following happens: either the primary velocity for all equal response dimensions (all elements of the primary velocity vector) are equal to zero or the maximum number of iterations of the algorithm has been reached [19].

Another parameter used in this algorithm is the maximum primary velocity. This parameter is the maximum value that can be used in the primary velocity vector. The primary velocity vector is used to select the movement direction for obtaining a neighbor, and this parameter prevents the move from increasing the primary velocity vector elements beyond a certain limit.

In [20], the authors tried to optimize the convergence speed of a GA with two changing points in the standard GA. After executing the crossover action, if the fitness value of the produced population is less than the average fitness or the best individual of the population, secondary preferential hybridization or mutation is also used after the primary mutation action.

Cruz-Chávez[21] proposed a hybrid genetic/annealing evolutionary algorithm for the independent task scheduling problem. The main purpose of this algorithm was to find the solution that minimizes the total runtime. GAs are weak for local searches, while SA is powerful for local searches. The authors combined these two methods to use both their abilities to search the problem space. The GA includes a stochastic population

generator, an elitism selection operator, and mutations and crossovers with the help of SA. Based on the fitness function, the selection operator selects the best half of the chromosomes in the population, the crossover is performed, and new children are produced for the next generation. Using a crossover leads to complete searches of the problem space. The iteration operation used as a mutation produces an optimized population, and a better population is found during the SA searching iteration. This process is repeated for each generation. It should be noted that thermal simulation techniques are performed on populations of individuals who have been run off.

In [22], some modifications of GAs are proposed to improve scheduling efficiency. These changes consist of the combination of the greedy algorithms, modified critical path (MCP) [23] and duplication scheduling heuristic (DSH) [24], with a GA to minimize the start time for tasks until, in the end, makespan is minimized. The algorithm also uses idle processor time. The algorithm has two fitness functions. The first function searches for chromosomes with the shortest makespan and the second function are designed to find the most appropriate chromosomes with respect to load balance.

In [25], aGA is presented in which chaotic variables are used instead of random variables for chromosome production. This leads to a distribution of solutions over the entire search space and avoids local minima, so that the best solutions and productions are obtained in a shorter time.

In [26], aGA is combined with the hill-climbing algorithm to repair chromosomes. This work modifies invalid individuals in each generation until they become valid individuals in a new population.

In [27], the GELS algorithm is used for resource reservation and independent task scheduling, so that in the objective function, if one resource can't execute a task within its specified deadline, the task is allocated to another resource for execution. Simulation results show that this algorithm decreases makespan compared to GAs. In previous methods, a decrease of the entire execution time was considered, while the number of tasks missing their deadlines and the load balance problem were not also considered. Our proposed algorithm tries to consider these three parameters simultaneously. Also, because a GA is weak in local searches, our proposed algorithm combines it with a local search algorithm to address this weakness. A combination of a GA and GELS is used because GELS searches the problem space well and finds better solutions compared to other local search algorithms such as hill-climbing and SA.

3 Scheduling Problem Description

The scheduling problem for independent tasks is an NP-hard problem that consists of N tasks and M machines. Each task should be considered to be processed by each of the M machines, so that the makespan is minimized. However, this only considers one of the QOS parameters, the time constraint, and ignores the cost. Therefore, we have introduced a deadline for every task such that each task should complete its execution before its deadline. Each task can be executed on only one resource and is not stopped before its execution is complete.

We use the expected time to compute the ETC matrix model described in [28]. Since our proposed scheduling algorithm is static, we assume that the expected execution time for each task i on each resource j has already been determined and has been set in the ETC matrix at ETC[i,j]. Also, the ready time (Ready [j]) for each machine j indicates when j has finished its previous task. The makespan is equal to the maximum complete time Completion_Time [i,j](Equation 2):

$$\text{makespan}=\text{Max}(\text{Completion\_Time}[i,j]) \quad \{1\le i\le N, 1\le j\le M\} \tag{2}$$

Completion_Time [i,j] is the time at which task i ends on resource j and is calculated according to Equation(3):

$$\text{Completion\_Time}[i,j]=\text{Ready}[j]+\text{ETC}[i,j] \tag{3}$$

The purpose of scheduling is to assign tasks to resources so that the final makespan and the number of tasks that miss their deadlines are minimized.

4 The Proposed Method (GGA)

The efficiency of genetic algorithms is highly dependent on how the chromosomes are represented. Here we use a simple method for representing chromosomes, in order to simplify the work of the crossover and mutation operators. Natural numbers are used for encoding the chromosomes. The numbers inside the genes are random numbers between 1 and M. The chromosome lengths are assumed to be task numbers. Figure 2 shows an example of the chromosome representation. For example, in the figure, task4—orT4—executes on Resource2.

T1	T2	T4	T3
1	3	2	4

Figure 2: Chromosome representation.

Initial Population: The initial population is created randomly. A source is selected randomly until the task being considered is executed on it. Each of the chromosomes produced is assumed to be a dimension of the problem (in fact, the problem's dimensions are just the neighbouring solutions that are obtained by changing the current solution). An initial random velocity is given to each of the problem's dimensions, ranging between one and the maximum velocity.

First Fitness Function: The basic purpose of task scheduling is to minimize makespan. This is the total time required until all of the input tasks complete their execution. It should be noted that this time should always be less than or equal to the maximum deadline among all the tasks. In our proposed method for task scheduling, a solution is more appropriate if in addition to decreasing makespan, it minimizes the number of tasks that miss their deadlines. Equation (4) calculates the first fitness function for each chromosome:

$$\mathrm{Fit}_1(\mathrm{ch}_i) = \frac{1}{\mathrm{makespan}(\mathrm{ch}_i)} + \frac{1}{\mathrm{miss\_task} * \mathrm{MD}} \quad (4)$$

Where miss_task is the number of tasks that have missed their deadlines in chromosome ch_i andMD is the maximum deadline for all tasks. As the equation shows, when the makespan and the number of tasks missing their deadlines are smaller, the fitness function value is greater, indicating the more promising chromosomes.

Second Fitness function: With respect to the basic purpose of task scheduling, minimizing makespan, several chromosomes may be found that have similar makespans but don't all balance the load among their resources. Hence, the second fitness function considers this factor after obtaining solutions with similar makespans, to find the most appropriate solution with respect to load balance.

If the execution time for resource R_j is E_time [R_j], the average execution time (avg) for all resources is as shown in Equation (5):

$$\mathrm{avg} = \sum_{j=1}^{\mathrm{num\_resources}} \left(\mathrm{E\_time}[R_j] \Big/ \mathrm{num\_resources} \right) \quad (5)$$

where num_resources is the number of resources. The load balance for resource i, $\mathrm{Cpu\_LB}_i$, can then be calculated with Equation (6):

$$\mathrm{Cpu\_LB}_i = \mathrm{makespan}/\mathrm{avg} \quad (6)$$

Equation (7)shows the second fitness function that considers the load balance:

$$\mathrm{Fit}_2(\mathrm{ch}_i) = \frac{1}{\mathrm{Cpu\_LB}_i} \quad (7)$$

Select an Operation: Before the mutation and crossover operators apply, the selection phase is first executed. In our proposed algorithm, we use the GELS algorithm instead of traditional genetic operators such as tournament, elitism, etc. These operators provide the possibility of creating the best solutions in each generation, but the GELS algorithm is used to select solutions because one chromosome may not initially have a good fitness value but turn out to be better after the mutation and crossover operations. Using the GELS algorithm, the two chromosomes that have a greater primary velocity are selected.

Crossover Operator: Our proposed algorithm uses a two- point crossover operator. Two points are selected randomly from chromosomes from the previous phase. Then all of the genes within these two points of the first and second chromosomes are removed.

Mutation Operator: A point on each chromosome from the previous phase is randomly selected and then changed to a random number between 1 and M.

Force Calculation: After applying the crossover and mutation operations, the gravitational force between the primary chromosome and the produced chromosome are calculated as in Equation (8). Then the gravitational force is added to the velocity of that dimension. This leads to no copying in the candidate population, if the produced chromosomes have worse fitness values than the primary chromosomes.

$$\mathrm{Force} = 6.672 * \left(\frac{\mathrm{Fit}_1(\mathrm{Candidate\_ch}_i)}{R^2} - \frac{\mathrm{Fit}_1(\mathrm{Current\_ch}_i)}{R^2} \right) \quad (8)$$

Terminating Conditions:The algorithm terminates when the primary velocity is equal to zero for all dimensions or the maximum number of algorithm iterations has been reached.

Algorithm1 shows the GGA pseudocode.

Algorithm 1 GGA Algorithm (pseudocode)

Input: Tasks Populations;
Output: Scheduled tasks based on Fit1 and Fit2;

```
1:  Generate K chromosomes to initialize the population
2:  Velocity_Vector[1..K]=Initial velocity for each
    Dimension();
3:  While (i<=max_iteration and Velocity_Vector[...]≠0)
    {
4:  /* select current_ch1 and current_ch2 such that the velocity
    is larger and generate two offspring, candidate_ch1 and
    candidate_ch2, by crossover and mutation*/
5:  If (Fit1(candidate_ch1) > Fit1(current_ch1) )
        current_ch1= candidate_ch1;
6:  If (Fit1(candidate_ch2)> Fit1(current_ch2))
        current_ch2= candidate_ch2;
7:  Calculate gravitational force between candidate_ch1 and
    candidate_ch2 using Equation (8)
8:  Update Velocity_Vector for each dimension by
    gravitational force of chromosome;
9:  }// end while
10: If many chromosomes with same Fit1exist
        Select Best chromosome using Fit2;
```

5 Performance Evaluation

Here, we explain the experimental descriptions.

5.1 Experimental Results

The GGA algorithm was implemented using Java software running under the Win XP operating system on a 2.66GHZ CPU with 4GB RAM. In our proposed algorithm, we assumed that the crossover rate CR =0.98 and the mutation rate MR =0.05.

The contents of the primary velocity vector for the chromosomes were randomly assigned. The results of

simulations comparing GGA with the GELS, GA, and GSA algorithms are shown in Figures 4, 5, and 6.

5.2 Experimental Results

Here, we have tested our work on various tasks; Generations and different fitness function orderly.

The diagram in Figure 3 shows a number of scheduled tasks ranging between 20 and 60 allocated to 20 resources using the comparison algorithms. As the figure shows, when the number of tasks increases, the makespan increases as well. The diagram shows that our proposed algorithm produces a smaller makespan than the other algorithms.

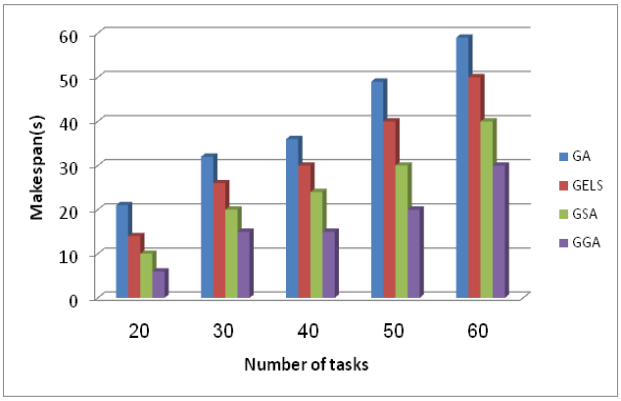

Figure 3: Comparison of make spans.

Figure 4 compares the algorithms for various numbers of iterations. It is clear that GGA, which is a combination of a globally searching GA and the local GELS algorithm, pays more attention to convergence velocity and optimization than the other algorithms, since unlike SA, the GELS algorithm doesn't have an absolute probability state.

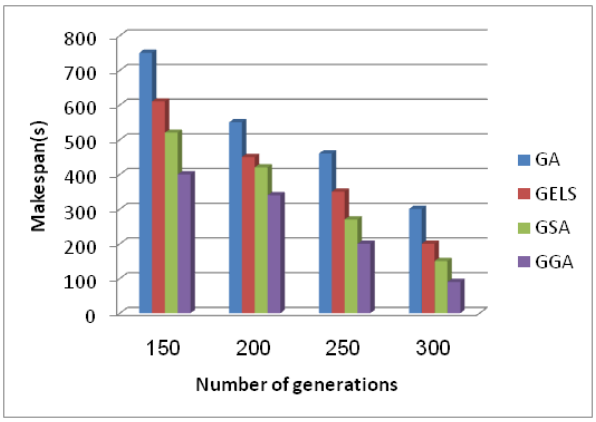

Figure 4: Comparison of evolutionary process for the different algorithms.

Figure 5 compares the algorithms with respect to the percentage of tasks that miss their deadlines. In this diagram, the fitness value is plotted against the rate of tasks missing deadlines. As the diagram shows, whenever the fitness value increases, the rate of tasks missing deadlines decreases. This means that the number of tasks missing deadlines decreases as a result of the completion of their makespan. The figure shows that fewer tasks miss their deadlines in the GGA algorithm than in the other algorithms.

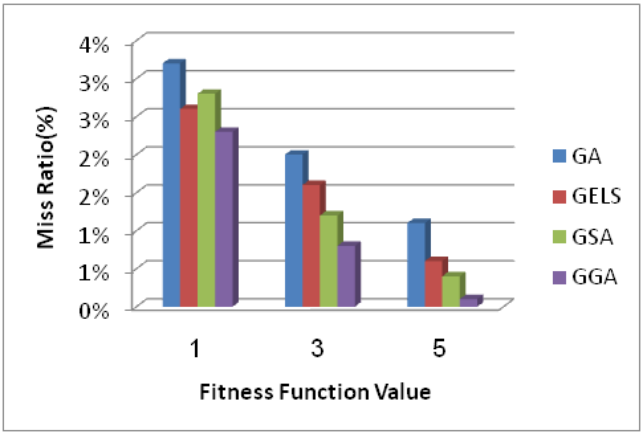

Figure 5: Comparison of the average missed deadline ratios for different fitness function values and algorithms.

6 Conclusions

This paper presented an algorithm for solving the grid task scheduling problem through a combination of a GA, which is a global search algorithm, and the GELS algorithm, which searches locally. The algorithm aims at minimizing makespan as well as the number of tasks that miss their deadlines. Local search algorithms such as hill climbing and SA always move to the solutions that have a better fitness function value, and they search the problem space randomly. Although the GELS algorithm shares the special behaviour of greedy algorithms, it doesn't always move directly to a solution with a better fitness function value but rather works by examining existing solutions. Although the GELS algorithm uses some random elements, it doesn't always move among them in the same way, which is why it doesn't stop with locally optimal solutions. By combining the advantages of the GELS algorithm and GAs, both the convergence velocity and the GA's identification of an optimal response are improved. We compared our proposed algorithm to other algorithms, and our simulation results showed that GGA produces smaller makespans than the other algorithms and also minimizes the number of tasks that miss their deadlines.

References

[1] J. Kołodzie and F. Xhafa, "Meeting security and user behavior requirements in Grid scheduling,"*Simulation Modeling Practice and Theory*vol.19, no. 1, pp. 213–226, 2011.

[2] W.T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye and D. Anderson, "A new major SETI project based on Project SERENDIP data and 100000 personal computers,"in *Proc. of the Fifth*

International Conference on Bioastronomy, no. 61, 1997.

[3] I. Foster, C. Kesselman, J. Nick and S. Tuecke, "the Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,"*Computer*, 35 (6),pp. 1-4, 2002.

[4] B.Yan-ping, Zh.Wei and Y.Jin-shou, "An Improved PSO Algorithm and Its Application to Grid Scheduling Problem,"in *International Symposium on Computer Science and Computational Technology ISCSCT '08*, 2008, pp. 352-355.

[5] M. Shojafar, S. Barzegar and M.R. Meybodi, "A new Method on Resource Scheduling in grid systems based on Hierarchical Stochastic Petri net," in *Proc. of third International Conference on Computer and Electrical Engineering (ICCEE 2010)*, 2010, pp. 175-180.

[6] M. Shojafar, Z. Pooranian, J.H. Abawajy and M.R. Meybodi, "An Efficient Scheduling Method for Grid Systems Based on a Hierarchical Stochastic Petri Net," *Journal of Computing Science and Engineering (JCSE)*, 7(1), pp. 44-52, 2013.

[7] M. Shojafar, S. Barzegar and M.R. Meybodi, "Msc.Thesis: Time Optimizing in Economical Grid Using Adaptive Stochastic Petri Net Based on Learning Automata,"M.s.c. Thesis, Islamic Azad University of Qazvin, Qazvin, Iran,September 2010.

[8] M. Shojafar, S. Barzegar, and M. R. Maybodi, "Time optimizing in Economical Grid Using Adaptive Stochastic Petri Net Based on Learning Automata", in *Proc. of International Conference on Grid Computing & Applications(GCA)*, WORLDCOMP, 2011, pp. 67-73.

[9] G. Falzon and M. Li, "Enhancing genetic algorithms for dependent job scheduling in grid computing environments,"*The Journal of Supercomputing*, Springer,62(1), pp. 290–314, 2012.

[10] Z. Pooranian, M. Shojafar, J.H. Abawajy and M. Singhal, "GLOA: A new Job Scheduling Algorithm for Grid Computing," *International Journal of Artificial Intelligence and Interactive Multimedia (IJIMAI)*, 2(1), pp. 59-64, 2013.

[11] W. Abdulal and S. Ramachandram, "Reliability-Aware Scheduling Based on a Novel Simulated Annealing in Grid," in *Proc. in Fourth International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 665-670. 2012.

[12] Z. Pooranian, M. Shojafar and B. Javadi, "Independent Task Scheduling in Grid Computing Based on Queen Bee Algorithm," *IAES International Journal of Artificial Intelligence (IJ-AI)*, 1(4), pp. 171-181, 2012.

[13] F. Xhafaa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future Generation Computer Systems*,26(4),pp. 608–621, 2010.

[14] Z. Pooranian, A. Harounabadi, M. Shojafar, J. Mirabedini, "Hybrid PSO for Independent Task scheduling in Grid Computing to Decrease Makespan," in *Proc. of International Conference on Future Information Technology,IPCSIT'11*, vol. 13, 2011, pp. 435-439.

[15] J. Holland, "Adaptation in Natural and Artificial Systems," University of Michigan Press, Ann Arbor, ISBN: 0-262-58111-6, 1975.

[16] C. Voudouris and T. Edward,"Guided Local Search. Technical Report CSM-247," Department of Computer Science, University of Essex, UK, August 1995.

[17] L. W. Barry, "Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction," Ph.D. Thesis, Florida Institute of Technology Melbourne, FL, USA, May 2004.

[18] S. R. Balachandar and K. Kannan, "Randomized gravitational emulation search algorithm for symmetric traveling salesman problem," *Applied Mathematics and Computation*, 192(2), pp. 413–421, 2007.

[19] J. Li, Y. Lou and Y. Shi, "An Optimization Algorithm Based on Binary Difference and Gravitational Evolution, "International *Journal of Computational Intelligence Systems*,5(3), pp. 483-493,2012.

[20] X. ZhangandW. Zeng, "Grid Workflow Scheduling Based on Improved Genetic Algorithm,"in *Proc. of International Conference on Computer Design and Applications (ICCDA 2010)*, 2010, pp. 270-273.

[21] M.Cruz-Chavez, A.Rodriguez-Leon, E.Avila-Melgar, F.Juarez-Perez, M.Cruz-Rosales and R. Rivera-Lopez,"Genetic-Annealing Algorithm in Grid Environment for Scheduling Problems,"*Security-Enriched Urban Computing and Smart GridCommunications in Computer and Information Science, Springer*, vol. 78, 2010,pp. 1-9.

[22] F.A. Omaraa and M.M. Arafa, "Genetic algorithms for task scheduling problem,"*Journal Parallel Distributed Computing*, Elsevier, 70(1), pp. 13-22, 2010.

[23] M. Wu and D.D. Gajski, "Hyper tool: A programming aid for message-passing systems,"*IEEE Transactions on Parallel and Distributed Systems*, 1(3), pp. 330-343,1990.

[24] A.P. Engelbrech, "Fundamentals of computational swarm intelligence,"John Wiley & Sons Inc.,2005.

[25] G. Gharoonifard,F. Moeindarbari, H. Deldari and A. Morvaridi, "Scheduling of scientific workflows using a chaos- genetic algorithm,"in *Proc. of International Conference on Computational Science ICCS2010*, 2010, pp. 1439-1448.

[26] A. Lifeng and T. Maolin, "QoS-Based Web Service Composition Accommodating Inter-Service Dependencies Using Minimal-Conflict Hill-Climbing Repair Genetic Algorithm," in *Proc. of Fourth IEEE International Conference on Science*, 2008, pp. 119-126.

[27] B. Barzegar, A.M. Rahmani and K. Zamanifar, "Gravitational Emulation Local Search Algorithm

for Advanced Reservation and Scheduling in Grid Systems," in *Proc. of First Asian Himalayas International Conference on (2009)*, 2009,pp. 1-5.

[28] T.D. Braun., H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.L. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen and R.F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems,"*Journal of Parallel and distributed Computing*, 61(6), pp. 680- 1983, 2001.

[29] S.S. Kim, J.H. Byeon, H. Liu, A. Abraham and Sean McLoone, "Optimal job scheduling in grid computing using efficient binary artificial bee colony optimization, *Soft Computing*," 17(5), pp. 867-882, 2013.

[30] H. Liu, A. Abraham and A. Hassanien, "Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm," *Future Generation Computing Systems, Elsevier Science*, 26 (8), pp. 1336-1343, 2010.

[31] H. Izakian, B.T. Ladani, A. Abraham and V. Snasel, "A Discrete Particle Swarm Optimization Approach for Grid Job Scheduling," International Journal of Innovative Computing, Information and Control, 6(9), pp. 4219-4233, 2010.

[32] H. Izakian, A. Abraham and V. Snasel, "Performance Comparison of Six Efficient Pure Heuristics for Scheduling Meta-Tasks on Heterogeneous Distributed Environments," *Neural Network World*, 19(6), pp. 695-710, 2009.

[33] H. Liu, A. Abraham and Z. Wang, "A Multi-swarm approaches to Multi-objective Flexible Job-shop Scheduling Problems," *Fundamental Informaticae Journal, IOS Press*, Netherlands, 95(4), pp.465-489, 2009.

[34] H. Izakian, A. Abraham and V. Snasel, "Metaheuristic Based Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems," *Sensors, Molecular Diversity Preservation International Switzerland*, 9(7), pp. 5339-5350, 2009.

[35] A. Abraham, H. Liu and M. Zhao, "Particle Swarm Scheduling for Work-Flow Applications in Distributed Computing Environments, Metaheuristics for Scheduling: Industrial and Manufacturing Applications," *Studies in Computational Intelligence, Springer Verlag, Germany*, ISBN 978-3-540-78984-0, pp. 327-342, 2008.

[36] A. Abraham, H. Liu, C. Grosan and F. Xhafa, "Nature Inspired Metaheuristics for Grid Scheduling: Single and Multi objective Optimization Approaches, Metaheuristics for Scheduling: Distributed Computing Environments," *Studies in Computational Intelligence, Springer Verlag, Germany*, ISBN: 978-3-540-69260-7, pp. 247-272, 2008.