Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 1 (2012) 1445–1454

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Scheduling of scientific workflows using a chaos-genetic algorithm

Golnar Gharooni-fard[a], Fahime Moein-darbari[b], Hossein Deldari[c], Anahita Morvaridi[d,*]

[a,b] *Computer Science department of Islamic Azad University, Mashhad Branch, Emamiyeh Blvd., Ghasem abad, Mashhad, Iran*
[c,d] *Computer Science department of Ferdowsi University, Azadi Square, Mashhad, Iran*

Abstract

The main idea of developing Grid is to make effective use of the computation power distributed all over the world. Economical issues are the most vital motivations of resource owners to share their services. This means that users are required to pay for access to services based on their usage and level of QoS they need. Therefore total cost of executing an application is becoming one of the most important parameters in evaluating QoS, which users tend to decrease.

Since, many applications are described in the form of dependent tasks, scheduling of these workflows has become a major challenge in grid environment. In this paper, a novel genetic algorithm called chaos-genetic algorithm is used to solve the scheduling problem considering both user's budget and deadline. Due to the nature of chaotic variables such as pseudo-randomness, ergodicity and irregularity, the evolutional process of chaos-genetic algorithm makes individuals of subgenerations distribute ergodically in the defined space and circumvents the premature of the individuals of traditional genetic algorithms (TGA). The results of applying chaos-genetic scheduling algorithm (CGS) showed greater performances of CGS compared to traditional genetic algorithm (TGS) on both balanced and unbalanced workflows.

Keywords: Grid computing; Chaos-genetic algorithms; Workflow scheduling; Deadline constraints; Budget constraints

1. Introduction

Grid computing is based on local grid computing which is basically, a kind of distributed computing (such as cluster computing and, point- to-point computing) which is capable of supporting diverse computing services. This has been made possible by the extra high speed internet and powerful processors that can execute middle wares without distracting computer's regular job. The main differences between Grid environment and traditional distributed systems are,

- There is no central control over the computers.
- General-purpose protocols are used.
- The Quality of Services is usually very high.

As the internet speed increases, the difference between two PCs working next to each other in a single building, or far from each other in a city or country gradually fades out. Therefore, users are able to execute their tasks on

* Anahita Morvaridi. Tel.: +98-0915-521-1840.
Email Address: a.morvaridi@krec.ir

1877-0509
doi:10.1016/j.procs.2010.04.160

geographically distributed sources. The main idea behind introducing Grid was that we could utilize the computation power in the same way that we use water, electricity and gas power. In other words, we are searching for a way to be able to connect to the tremendous computational power of the whole universe, where the costs are directly dependent to the amount of energy being utilized. This has led to the idea of "economical Grid" which adds the concept of considering execution cost issues in computation algorithms. Therefore, taking into account the main objective of increasing the performance, our focus is no longer limited to raising the speed of the computation but also to reducing its execution cost.

In general, we can say that traditional models for scheduling Grid are pretty frail. Considering Grid characteristics, a user may request an application that can be executed on the other side of the world, where network properties such as bandwidth, management policies, computational capabilities and etc are totally different. Therefore Grid scheduling has turned to be a major challenge. Here's a list of the most important challenges of scheduling in Grid environment:

- Sources are usually shared between the users so there may be a competition among them.
- The scheduler is not in control of the sources.
- The number of available sources is constantly changing.
- Sources are located on different management sites.
- Sources are heterogeneous.
- Most of the workflow applications are data-centric and therefore need a large amount of data transfer between two sites.

In this paper we investigate the problem of scheduling workflows considering the QoS constraints. Since this problem is an NP-complete one, we proposed a meta-heuristic algorithm based on genetic algorithms to solve the workflow scheduling problem with the objective of minimizing time and cost of the execution.

The cost of a service is normally related to the quality of the service it provides. Generally, service providers charge more money in response to higher quality of service. In addition, users may not always desire to complete workflows earlier than they require. Cheaper services with lower QoS that is sufficient to meet the user's requirements are sometimes preferred. Therefore, a trade off between the time and monetary cost needs to be considered.

Given this motivation, we suggest a method considering time and cost simultaneously, when scheduling a workflow execution. The remainder of the paper is organized as follows. We introduce related work in the next section. Then a general overview of the scheduling problem is explained, followed by defining the basic concepts used in our algorithm. Our proposed chaos-genetic algorithm is presented in section 4. Experimental details and simulation results are presented in section 5. Finally, we conclude the paper with directions for further work in the last section.

2. Related work

Several heuristics have been proposed to solve the workflow scheduling problem. Generally, scheduling algorithms can be classified into two major groups, in view of their main objectives. First, a group of works that only attempt to minimize workflow execution time, without considering user's budget. *Minmin*, which sets the highest priority to tasks with the shortest execution time, and *Maxmin*, which sets the high priority to the tasks with the long execution times are two major heuristic algorithms employed for scheduling workflows on Grids. *Sufferage*, is another heuristic algorithm which sets high scheduling priority to tasks whose completion time by the second best resource is far from that of the best resource which can complete the task at earliest time. These algorithms have been used to schedule *EMAN* bio-imaging application in [1].

Blythe et al [2] developed a workflow scheduling algorithm based on Greedy Randomized Adaptive Search Procedure (*GRASP*) [3] and compared it with *Minmin* in different scenarios. In [4], another heuristic algorithm based on genetic algorithms was proposed which takes into account the information of the entire workflow. Another workflow level heuristic is a Heterogeneous-Earliest-Finish-Time (*HEFT*) algorithm proposed by Wieczorek et al. in [5]. Second, a group of works which address scheduling problems based on user's budget constraints. *Nimrod-G* [6] schedules independent tasks for parameter-sweep applications to meet user's budget. More recently, Tsiakkouri et al [7] developed scheduling approaches, *LOSS* and *GAIN*, to adjust a schedule which is generated by a time optimized heuristic and cost optimized heuristic to meet user's budget constraints. Our aim is to introduce a new

method based on genetic algorithms to solve the scheduling problem considering the budget and deadline of entire network.

3. Problem description

A workflow application can be modelled as a Directed Acyclic Graph (DAG). There is a finite set of tasks T_i (i = *1,2, …, n)* and a set of directed arcs of the form *(* T_i *,* T_j *)*, where T_i is the parent task of T_j , and T_j is the child of T_i. A child task can never be executed unless all of its parent tasks have been completed. Let B be the cost constraint (budget) and D be the time constraint (deadline), specified by the user's workflow execution.

Let m be the total number of services available. There's a set of services $S_i^j (i = 1,2, \dots, n , j = 1,2, \dots, m_i , m_i \leq m)$ capable of executing taskT_i, but each task can only be assigned for execution one of these services. Services have varied processing capability delivered at different prices. We denote t_i^j as the processing time, and c_i^j as the service price for processing T_i on service S_i^j.

The scheduling problem is to map every T_i onto a suitable S_i^j in order to improve the execution time and cost of a workflow according to the user's budget and deadline. In the next section, we'll introduce the main concepts used to design the algorithm.

3.1. Genetic Algorithms

Genetic Algorithms were introduced by John Holland in early seventies as a special technique for function optimization. Genetic algorithms are based on the biological phenomenon of genetic evolution. The basic idea is as that the genetic pool of a given population potentially contains the solution, or a better solution, to a given adaptive problem. This solution is not active because the genetic combination on which it relies is split between several subjects. Only the association of different chromosomes can lead to the solution. During reproduction and crossover, new genetic combinations occur and, finally, a subject can inherit a good gene from both parents. The algorithm operates in an iterative manner and evolves a new generation from the current generation by application of genetic operators. A new generation is created by first increasing the population by random individual solutions and then selecting a constant number of solutions based on their fitness values [8].

Therefore given a clearly defined problem to be solved and strings of candidate solutions, a simple GA works as follows:

1. Initialize the population.
2. Calculate fitness for each individual in the population.
3. Reproduce selected individuals to form a new population.
4. Perform crossover and mutation on the population.
5. Loop to step 2 until some condition is met.

In some GA implementations, operations other than crossover and mutation are carried out in step 4. Crossover, however, is considered by many to be an essential operation of all GAs. Termination of the algorithm is usually based either on achieving a population member with some specified fitness or on having run the algorithm for a given number of generations.

3.2. Chaos

Chaos is a none-periodic, long-term behaviour in a deterministic system that exhibits sensitive dependence on initial conditions. Edward Lorenz irregularity in a toy model of the weather displays first chaotic or strange attractor in 1963. It was mathematically defined as randomness generated by simple deterministic systems. A deterministic structure can have no stochastic (probabilistic) parameters. Therefore chaotic systems are not at all equal to noisy systems driven by random processes. The irregular behaviour of the chaotic systems arises from intrinsic nonlinearities rather than noise.

In general, the most important defining property of chaotic variables is Sensitive dependence to Initial Conditions (SIC), which requires that trajectories originating from very nearly identical initial conditions diverge at an exponential rate. Pseudo-randomness and ergodicity are other dynamic characteristics of a chaotic structure [9]. The

latter ensures that the track of a chaotic variable can travel ergodically over the whole space of interest. The variation of the chaotic variable has a delicate inherent rule in spite of the fact that it looks like a disorder.

3.3. Chaos-Genetic Algorithm

Recently, the idea of using chaotic systems instead of random processes has been noticed in several fields. One of these fields is optimization theory. In random-based optimization algorithms, the role of randomness can be played by a chaotic dynamics. Experimental studies assert that the benefits of using chaotic signals instead of random signals are often evident although it is not mathematically proved yet [10]. For example in genetic algorithms, chaotic sequences increase the value of some measured algorithm-performance indexes with respect to random sequences.

In this paper a Chaos-Genetic Scheduling algorithm, CGS, is proposed that combines the concept of chaos with genetic algorithms when looking for an optimal solution, in order to possess a joint advantage of GA and the chaotic variable [11]. Firstly, CGS takes the advantages of the characteristics of the chaotic variable to make the individuals of subgenerations distributed ergodically in the defined space and thus to avoid the premature convergence of the individuals in the subgenerations. Secondly, CGS also takes the advantage of the convergence characteristic of GA to overcome the randomness of the chaotic process and hence to increase the probability of finding the global optimal solution.

The idea of combining chaos with Genetic Algorithm has also been studied in other computer-related fields. In [12] a chaos-genetic based approach is proposed in order to solve the Network-on-Chip mapping problem. In the field of neural networks, chaos search is used to accompany GA in order to overcome the weakness of Traditional Genetic Algorithm (TGA) [13]. In [14] a chaos-genetic algorithm based on the chaos optimization algorithm (COA) and genetic algorithm, is presented to overcome premature local optimum and increase the convergence speed of genetic algorithm. Simulation results indicate that the Chaos GA can improve convergence speed and solution accuracy, in all the literature mentioned above.

4. The proposed algorithm

For a workflow scheduling problem, a feasible solution is required to meet several conditions. A task can only be started after all its predecessors have completed, every task appears once and only once in the schedule, and each task must be allocated to one available time slot of a service capable of executing the task.

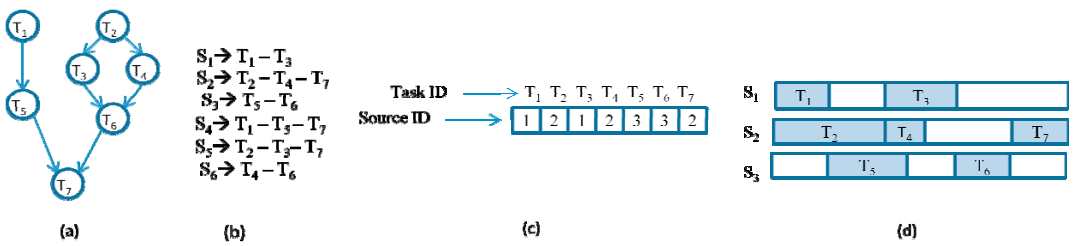


Fig.1. Illustration of problem encoding, (a) sample workflow, (b) set of source-to-task assignments, (c) an example of a one-dimensional chromosome, (d) execution order of the sample chromosome.

Each individual in the population represents a feasible solution to the problem, and consists of a vector of task assignments. Each task assignment includes four elements (task ID, service ID, start time, end time). The first two parameters identify to which service each task is assigned. Since involving time frames during the genetic operation may lead to a very complicated situation [15], in this work we ignore the time frames. Therefore, the operation strings (chromosomes) encode only the service allocation for each task and the order of the tasks allocated on each service. Different execution priorities of such parallel tasks within the workflow may impact the performance of workflow execution significantly. For this reason, the solution representation strings are required to show the order of task assignments on each service in addition to service allocation of each task. As it is also used in [15], we create a 2D string to represent a schedule as illustrated in Fig.1. One dimension represents the numbers of services whereas

the other dimension shows the order of tasks on each service. Two-dimensional strings are then converted into a one-dimensional string for genetic manipulations.

As stated earlier, the problem is to schedule a workflow execution considering both time and user budget constraints. The first decision to be made is how to represent the solution. Fig.1 shows an example of an individual in the initial population. Initializing the population is another important issue, which is usually done randomly. Therefore a random number generator is used to produce values between 1 to *n*. For each task, these random values are chosen from the sources that are capable of executing that task. The length of the chromosome depends on the number of tasks in the workflow.

A chaotic mapping operator is then applied to the initial population generating a new chaotic population. The evolution process of the chaotic variables could be defined through the following equation:

$$cs_i^{(k+1)} = 4cs_i^{(k)}\left(1 - cs_i^{(k)}\right), \qquad i = 1, 2, \dots, m \tag{1}$$

in which cs_i is the *i*-th chaotic variable and *k* and *k+1* denote the number of iterations. Note that values of cs_i are distributed in the range of (0,1). The chaotic mapping operator works as follows:

1. Divide the interval (0,1) to *m* equal sub-intervals (*m* denotes the number of resources capable of executing a special task).
2. The value of each gene in the first randomly produced population is mapped to new values of cs_i in the range of (0,1).
3. These values of $s_i^{(1)}$, *i = 1,2,..., n* are linearly mapped using the operator

$$\frac{s_i^{(1)}}{m_i} = cs_i^{(1)} \tag{2}$$

where m_i is the total number of resources capable of executing T_i.

4. The next iteration chaotic variables $cs_i^{(2)}$, will be produced through applying Equation.1 to the values of $cs_i^{(1)}$, generated in the previous step.
5. The chaotic variables $cs_i^{(2)}$, are then used to produce $s_i^{(2)}$, using

$$s_i^{(2)} = \left\lceil cs_i^{(2)} \times m_i \right\rceil \tag{3}$$

Thus, we can continue to produce the values of $s_i^{(k)}$ for each chromosome, through the operators defined in (1) - (3).

At this stage, the fitness of all 20 individuals is evaluated. The fitness value is often proportional to the output value of the function being optimized according to the given objectives. As the goal of scheduling is to improve the performance of a workflow execution by minimizing the time and cost, the fitness function separates evaluation in two parts [15]: cost-fitness and time-fitness.

For the budget constrained scheduling, the cost-fitness component produces results with less cost. The cost fitness function of an individual I is defined by:

$$F_{cost}(I) = \frac{c(I)}{B} \tag{4}$$

where *c(I)* is the sum of the task execution cost and data transmission cost of *I* and *B* is the budget of the workflow.

For the budget constrained scheduling, the time-fitness component is designed to produce individuals that satisfy deadline constraint. The time-fitness function of an individual *I* is defined by:

$$F_{time}(I) = \frac{t(I)}{D} \tag{5}$$

where *t(I)* is the completion time of *I*, *D* is the deadline of the workflow. The final fitness function combines the two parts and it is expressed as:

$$F(I) = \begin{cases} F_{cost}(I) + F_{time}(I), & if\ F_{cost}(I) > 1\ or\ F_{time}(I) > 1 \\ \frac{c(I)}{maxcost} \times \frac{t(I)}{maxtime}, & otherwise \end{cases} \quad (6)$$

where *maxcost* is the most expensive solution of the current population and *maxtime* denotes the largest completion time of the current population.

Elitism is incorporated into the algorithm by transferring the single fittest individual directly to the next generation. Crossover is performed on randomly selected individuals according to the idea that it may result in an even better individuals by combining the two fittest ones [10]. The crossover operator used in this algorithm is a basic two-point crossover which works as follows:

1. Two random parents are chosen in the current population.
2. Two random points are selected from the schedule order of the first parent.
3. All tasks between these two points are chosen as successive crossover points.
4. The locations of all tasks of the crossover points between the two parents are exchanged.
5. Two new offsprings are generated by combining task assignments taken from two parents.

Fig.2. shows an example of the process explained above.

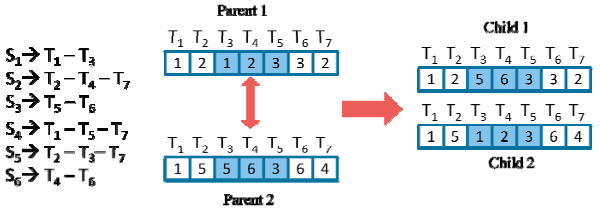


Fig.2. Crossover operation

Finally, a constant mutation rate (0.05) is applied in our proposed algorithm. Mutation aims to reallocate an alternative service to a task in an individual. An example of the mutation process is illustrated in fig.3. It is implemented as follows:

1. A task is randomly selected in a chromosome.
2. An alternative service which is also capable of executing the task is randomly selected to replace the current task allocation.

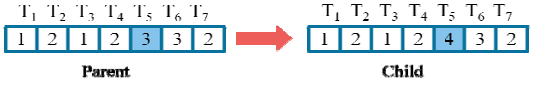
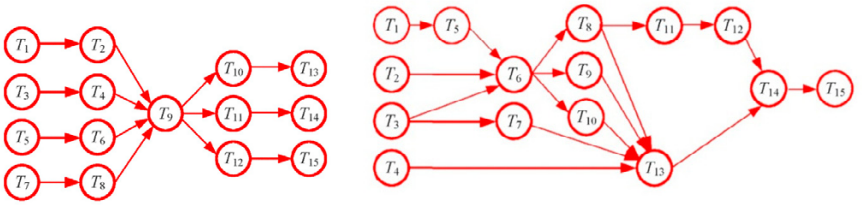


Fig.3. Mutation operation

The new population is now ready for another round of chaotic mapping, crossover, and mutation, producing yet another generation. So the initial population is replaced by these newly generated individuals. Obviously, more generations are produced until the stopping condition (a maximum number of generations *k*) is met. The fittest chromosome is thus returned as a solution.

5. Experimental results

According to workflow projects, workflow applications can be categorized as either balanced structure or unbalanced structure. Our proposed algorithm is applied to examples of both balanced and unbalanced structures. We use two common workflow applications for our experiments: A balanced application (fMRI workflow shown in fig.4 (a)) and an unbalanced structure (DNA workflow, shown in fig.4 (b)).



(a) (b)

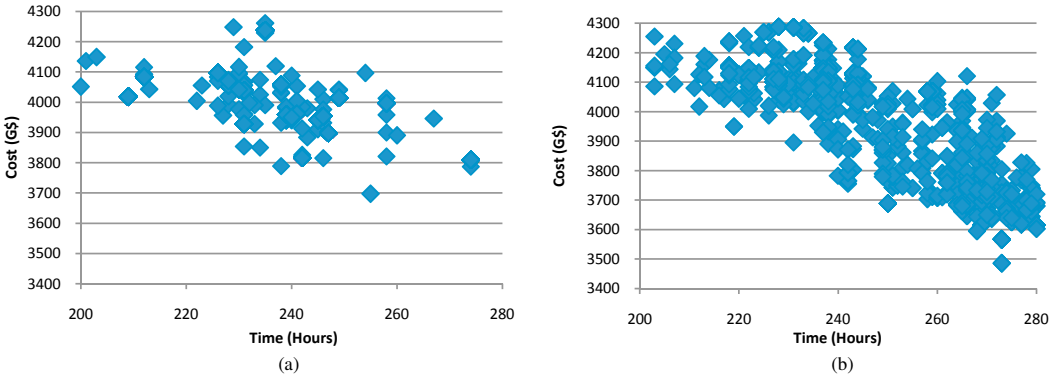
Fig.4.(a) A balanced workflow (fMRI), (b) An unbalanced workflow (DNA)

The two metrics used to evaluate our algorithm (CGS), are execution time and cost. Table 1 show service speed and corresponding price (time and cost) for executing T_1 on different sources, for fMRI workflow. First column of the table denotes the number of sources capable of executing T_1. For example, in fMRI, T_1 can be executed on five sources S_1-S_5.

Table 1. Data samples for executing T_1 in fMRI workflow

Source ID	Time	Cost
1	14	150
2	11	144
3	10	151
4	16	119
5	8	157

The following parameter settings are the default configuration for simulating both Genetic Algorithm and Chaos-Genetic Algorithm. Population size of 10 normal chromosomes followed by 10 chaotic chromosomes, crossover probability of 0.98 and mutation probability of 0.05.In order to be able to evaluate the results of our proposed algorithm (CGS), we also implemented a traditional genetic algorithm to solve the workflow scheduling problem. Since GA is a stochastic search algorithm, each of the experiments was repeated 10 times and the average values are used to report the results.



(a) (b)

Fig.5. Distribution of individuals when executing (a) TGS (b) CGS on DNA workflow

As mentioned earlier, the major characteristic of CGS is that it prevents the premature convergence of the individuals of TGS and thus increases the probability of finding better solutions. In order to illustrate the distribution of the individuals in our problem, we run our algorithm (CGS) to execute DNA workflow and compare the results with TGS. The distribution of the individuals of total 100 generations is illustrated in fig.5.

As it is clear in Fig.5 (b), the individuals of subgenerations generated by CGS are almost evenly scattered over the defined space and do not concentrate to the centre of the space anymore (see fig.5 (a)). Although the same numbers of solutions are considered in plotting both figures, the reason they look less in the case of TGS is that they are mostly very close to each other, and therefore the differences are not really clear in Fig.5 (a).

In order to evaluate algorithm on reasonable budget and deadline constraints, we also implemented a Traditional Genetic algorithm for scheduling workflow applications (TGS), so that it would be possible to compare the results obtained from CGS with the ones gained from TGS, for the same workflow applications.

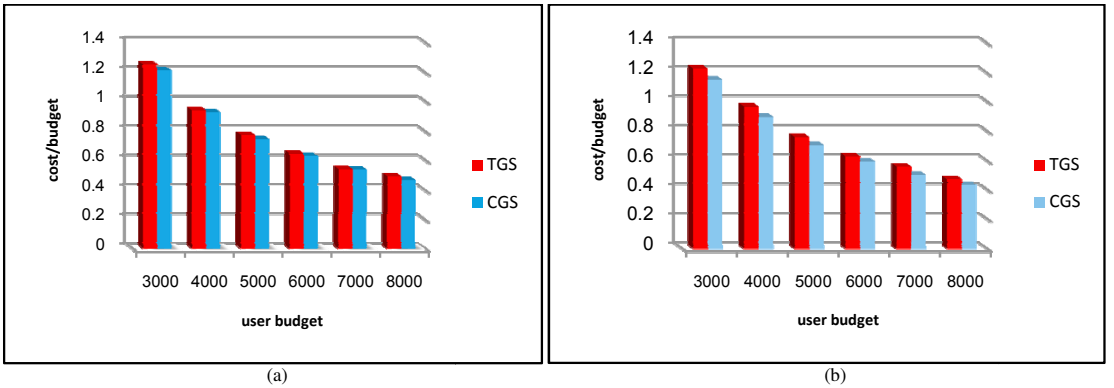


Fig.6. Comparison between the execution cost of TGS and CGS on balanced (fMRI) and unbalanced (DNA) structures

In Fig.6 (a) the results are obtained under the assumption of D = 220(H) and in Fig.6 (b) we assume, D = 240(H). The values of these assumptions are made based on [15]. The values in vertical axis are the result of the total cost divided by the user budget constraint, starting from G$3000 to G$8000. We observe that both TGS and CGS cannot satisfy the low budget constraint (about G$3000), and TGS shows the worst results in both applications. However, results are gradually improved under medium budget constraints (about G$5000). Obviously, the descending behaviour of the diagram shows that as the budget increases, it'll be easier for the algorithms to meet the user budget constraint. On the other hand, considering the differences between two approaches, it's obvious that TGS takes much longer to complete even when the budgets are high. Therefore, CGS shows better performance compared to TGS in both applications.

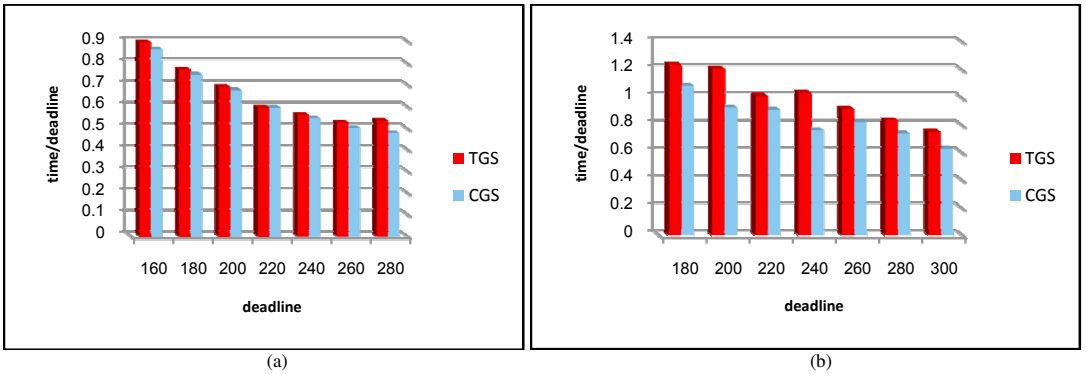


Fig.7. Comparison between the execution time of TGS and CGS on balanced (fMRI) and unbalanced (DNA) structures

Fig.7 illustrates the comparison between the execution times of the two algorithms with the medium budget of 5000. We change the user deadline values from 180(H) to 300(H) for DNA and 160(H) to 280(H) for fMRI, since the latter is a balanced workflow and takes less time to complete. It can be seen that TGS takes longer to complete in most of the conditions. The differences are better observed in the unbalanced workflow structure (see fig.7 (b)).

In all the above illustrations, there may be states where CGS and TGS show similar results (for instance in fig.6 (a) under budget constraint of 7000). These are the conditions where, TGS solutions are not trapped in a local optimum so it works as well as CGS in finding the good results for a given problem. In those conditions, CGS does not do any good in saving the suitable solutions. In the rest of the states though, TGA, is stocked somewhere in a local optimum (as it usually does), which prevents the algorithm from producing better possible results. Our chaos-genetic algorithm (CGS), takes the advantages of the characteristics of the chaotic variable to make the individuals of subgenerations distributed ergodically in the defined space and thus to avoid from the premature of the individuals in the subgenerations. It also takes the advantage of the convergence characteristic of TGA to overcome the randomness of the chaotic process and hence to increase the probability of finding the global optimal solution.

7. Conclusion and future works

In this work we introduce a novel chaos-genetic based algorithm that uses chaotic sequences instead of random processes in traditional genetic algorithms. We evaluate our approach by employing it to both balanced and unbalanced workflow structures. The results show better performances of Chaos Genetic Scheduling (CGS) algorithm in both cases, when compared with Traditional Genetic (TGS). The reason is that, chaos-genetic algorithm uses the characteristics of chaotic variables in scattering the solutions among the whole search space and thus avoids the premature convergence of the solutions and produces better results within a shorter time.

We will be further enhancing our scheduling algorithm by considering other QoS properties such as reliability. The performance of the algorithm can be improved by using the properties of chaotic sequences in other random decisions made in traditional genetic algorithms such as specifying crossover points. We can also apply other one-dimensional chaotic maps instead of Logistic map and compare the performance of our algorithm to find out which one works best for our scheduling problem.

Acknowledgments

This work is supported by the Iranian Telecommunication Research Center (ITRC) and the Young Researchers Club.

References

1. A. Mandal et al., "Scheduling Strategies for Mapping Application Workflows onto the Grid", *In IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, 2005.
2. J. Blythe et al., "Task Scheduling Strategies for Workflow-based Applications in Grids", *In IEEE International Symposium on Cluster Computing and Grid (CCGrid)*, 2005.
3. T. A. Feo and M. G. C. Resende, "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, 6:109-133, 1995.
4. R. Prodan and T. Fahringer, "Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study", In *20th Symposium of Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005. ACM Press.
5. M. Wieczorek, R. Prodan and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment", Special Issues on scientific workflows, *ACM SIDMOD Record*, 34(3):56-62, ACM Press, 2005.
6. R. Buyya, J. Giddy, and D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications", In 2nd *Workshop on Active Middleware Services (AMS 2000)*, Kluwer Academic Press, August 1, 2000, Pittsburgh, USA.
7. E. Tsiakkouri et al., "Scheduling Workflows with Budget Constraints", In *the CoreGRID Workshop on Integrated research in Grid Computing*, S. Gorlatch and M. Danelutto (Eds.), Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, Nov. 28-30, 2005, pages 347-357 .
8. Melanie, Mitchell. 1998. An Introduction to Genetic Algorithms, A Bradford Book The MIT Press, Cambridge, Massachusetts. London England.
9. Peter Stavroulakis, 2006, *Chaos Application in Telecommunications*, Taylor & Francis.
10. M.Bucolo, R.Caponetto, L.Fortuna, M.Frasca, and A.Rizzo, 2002, Does chaos work better than noise? *IEEE Circuits and Systems Magazine 2 (3)*, 4-19.
11. X.F.Yan , D.Z.Chen and S.X.Hu, 2003, Chaos-genetic algorithms for optimizing the operating conditions based on RBF-PLS model" *Elsevier Computers and Chemical Engineering* , 1393-1404.
12. F. Moei-darbari, A. Khademzadeh, and G. Gharooni-fard, "Evaluating the performance of chaos genetic algorithm for solving the Network-on-Chip mapping problem", in *proc. IEEE International Conference on Computational Science and Engineering*. Vancouver, Canada. vol. 2, pp.366-373, August 2009.

13. Y. Yong, S. Wanxing, and W. Sunam, "Study of chaos genetic algorithms and its application in neural networks", in proc. *IEEE TENCON'02*, pp. 232- 235, November 2008.
14. C. Cheng, W. Wang, D. Xu, and K.W. Chau, " Optimizing hydropower reservoir operation using hybrid genetic algorithm and chaos", *Water Resources Management*, Vol. 22, No. 7, 2008, pp 895-909.
15. J. Yu and R. Buyya, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, *ScientificProgramming*,14:217-230, 2006.