Computational Sciences and Engineering 3(2) (2023) 177-188

University of Guilan

Computational Sciences and Engineering

journal homepage: https://cse.guilan.ac.ir/

An Enhanced Genetic Algorithm for Task Scheduling in Heterogeneous Systems

Saeed Mirpour Marzuni [a], Javad Vahidi [b,*]

[a] Department of Electrical & Computer Engineering, University of Science and Technology of Mazandaran, Behshahr, Iran
[b] Department of Computer Science, Iran University of Science and Technology, Tehran, Iran

A R T I C L E I N F O

Article history:
Received 17 January 2024
Received in revised form 4 April 2024
Accepted 4 April 2024
Available online 4 April 2024

Keywords:
Genetic Algorithm,
Distributed Processing,
Task Scheduling

A B S T R A C T

Generally, jobs are divided into smaller portions, in parallel and according to distributed processing, and each portion is called a task. Each task can execute dependently or independently. When introducing heterogeneous systems, it is desirable that tasks can run on these systems. Since it is advantageous that tasks running on heterogeneous systems are completed faster, the optimization of task scheduling is of great importance. Actually, task scheduling problems in heterogeneous systems are NP-hard and it is a crucial issue. In such problems, Directed Acyclic Graphs (DAGs) can be used as task graphs to be scheduled on heterogeneous systems. The proposed method presents a genetic algorithm with new operators and final scheduler to be scheduled on heterogeneous systems. The practicality and convergence of the algorithm are proved by Markov's chain theory. The findings reveal that the currently proposed algorithm is more efficient in comparison to previously presented ones and also has a better make span. Moreover, it is concluded that the Enhanced Genetic Algorithm (EGA) achieves the solution faster in early generations.

1. Introduction

Scheduling is an important complex problem in heterogeneous systems. When several tasks are run on heterogeneous machines, the scheduling problem emerges as an important issue. Several studies have been conducted on task scheduling in heterogeneous systems which are connected to each other via a high speed network. In these systems, a job is divided into tasks of which each can be run in a separate system so as to process it faster to the finish. However, care should be taken to notice the parallel potential of the task; otherwise, not only will the job not end sooner, but it will also finish later due to communication overhead.

[*] Corresponding author.
E-mail addresses: jvahidi@iust.ac.ir (J. Vahidi)

https://doi.org/10.22124/cse.2024.26504.1072

Generally, great applications consist of smaller tasks so they are processed in a parallel. These small tasks are often dependent, meaning that the task results must be executed [1]. One of the issues mentioned in this field is task scheduling in heterogeneous systems. These systems are those having different resources working together to fulfill a job. The efficiency of the parallel execution of applications in heterogeneous systems depends on the way in which the tasks of an application are selected for scheduling. The purpose is to minimize the total response time or make span [2]. The reason why the task scheduling problem in heterogeneous systems is more complicated than that of homogenous systems is because there are various processors whose run time is different for each task on each processor. In addition to this difference in run time, there is significant variance in the communication overhead among various processors. The task scheduling problem can be presented on a Directed Acyclic Graph (DAG), in which each vertex of the graph illustrates one task and the edges indicate the priority between tasks [1]. The edge weight also shows the cost of the communication among processors. For instance, if E(i,j)=10, then task *i* must be run before task *j* and, if these two tasks run in two different processors, the communication cost will be 10.

Task scheduling problems are divided into two categories [2]: static scheduling, in which all the information about tasks is prior specified, including the run-time, communication cost, and priority of each task; and dynamic scheduling, in which information is not available and decisions are made at run-time. Researchers have proposed various algorithms to solve static scheduling problems. These are categorized as deterministic (heuristic-based algorithm) and nondeterministic (random search-based algorithms). Deterministic methods are divided into three types: list scheduling, cluster scheduling, and task duplication scheduling.

The nondeterministic scheduling methods consist of various categories [3]: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Tabu Search (TS), Simulated Annealing, Random Search, and the Genetic Algorithm (GA). The current paper proposes an algorithm for static task scheduling using random search and genetic algorithms as non-deterministic method.

In this paper, we proposed a new genetic algorithm with final scheduler that can schedule tasks efficiently. Our algorithm, using genetics algorithm and final scheduler, can efficiently schedule the tasks in heterogeneous systems.

The remainder of the present paper is organized as follows: In the next section the literature in this field is reviewed. Section 3 states the problem and identifies the symbols needing exploration. Section 4 describes the suggested algorithm in detail, the convergence of which is checked by Markov's model. In Section 5, the experiments and the obtained results are then analyzed and compared with other algorithms. Finally, the conclusion is presented in the last section.

2. Related Work

The task scheduling problem can be formulated as a search for the optimum assignment of a set of tasks to a set of processors so that the time of the last executed task will be minimized [1]. The NP-completeness of DAG scheduling for homogenous systems is presented in [4], [5], and [6]. Accordingly, researchers attempt to solve this problem utilizing heuristic methods to achieve a suitable scheduler. A heuristic-based algorithm searches a solution space where some feasible solution has not been considered [7,8]. Most heuristic scheduling belongs to a scheduling list category. When searching in a scheduling list, algorithms are divided into two phases. In the first phase, each task is assigned to a priority queue and then this is added to a list of pending tasks so as to lower the priority based on a specific scale. In the second phase, each task with the highest priority is selected and assigned to a processor. One of the best heuristics-based lists in a heterogeneous system is the HEFT [9].

Canon et al. [10] conducted a study to compare 20 scheduling heuristics and their findings revealed that HEFT was one of the best heuristics in terms of robustness and schedule length. One of the disadvantages of list scheduling algorithms based on heuristics is that their performances are greatly affected by heuristics. Therefore, a constant result for task scheduling might not be produced [1]. The other heuristic-based category is the cluster, which is generally designed for homogenous systems and is composed of clusters of tasks assigned to processors. For heterogeneous systems, the CHP [11] and Triplet [12] algorithms are designed for but limited to systems with a higher degree of heterogeneity [2]. Frequent heuristics may produce the lowest make span, but they feature two shortcomings: higher time complexity and task execution frequency. Genetic algorithms are widely utilized to solve problems with task scheduling. These algorithms differ in coding, implementation of genetic operators, and evaluating methods. In this paper, a genetic algorithm is proposed at first. Its result is several suggestions for task scheduling. After that, the final scheduler is applied for selecting the best suggestions. The proposed genetic algorithm is then modeled with the Markov Chain and, finally, the convergence of the algorithm is proved by it.

3. Problem Statement

In the current study, the task scheduling problem is solved by EGA. The underlying assumption is that there is a set of processors which are connected via a network. Task scheduling is responsible for mapping tasks to processors in the system and determining the order of their execution in each processor [13]. Scheduling problems are divided into two categories: static and dynamic. The aim is assessing the static task scheduling problem so there is enough information on the task priority and execution time of each task on each processor. In addition, it is necessary to be aware of any wasting of time between two tasks which are interdependent and executed in two different processors (i.e. the data movement time between two processors in two dependent tasks).

3.1. Dag Modeling

DAG is a directed graph that has no directed cycles (or no path with the same initial and final vertex). In ***Figure 1***, a directed acyclic graph, G=(V, E), is shown in which V is the set of vertices and E represents the edges. This kind of directed acyclic graph is utilized for the task scheduling problem to show the dependency of tasks. In each graph, each vertex displays one task in each problem and the edges show the priority between tasks. For instance, E(i,j) goes from *i* to *j* indicating that task *i* has more priority than task *j* or, in other words, task *i* must be executed before task *j*.

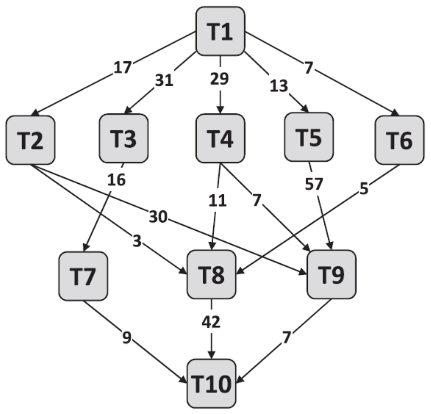
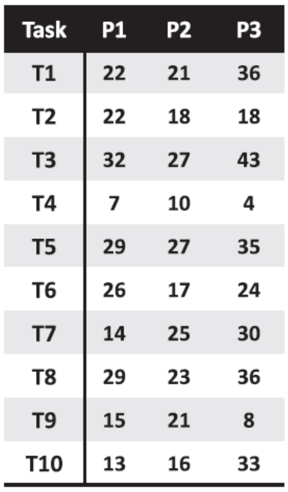


Figure 1. A DAG for the task scheduling problem [2]

Each edge has a weight of W_{ij} that specifies the time consumed to move data between two processors. For example, W_{12}=17 means that if task T_1 is executed in a processor and then task T_2 in the other processor, the required cost to move data will be 17; however, if the two tasks are executed in one processor, the cost will be zero. In addition to the DAG, a matrix of M*N is illustrated, in which M is the number of tasks that should be executed and N is the number of existing processors in the heterogonous system. As mentioned before, the speed of each one of the processors in the heterogeneous system is different, so the execution time of the tasks on each processor will take a different amount of time, as shown in the matrix in ***Figure 2.***



Task	P1	P2	P3
T1	22	21	36
T2	22	18	18
T3	32	27	43
T4	7	10	4
T5	29	27	35
T6	26	17	24
T7	14	25	30
T8	29	23	36
T9	15	21	8
T10	13	16	33

Figure 2. Required time for running each task on each processor [2]

4. Enhanced Genetic Algorithm

The current study proposes a task scheduling problem by applying a genetic algorithm. The genetic algorithm was introduced by Holland in 1975 [14] and is considered as one of the random search algorithms conceived from nature. Genetic algorithms are mostly used in the optimization of problem solving. Naturally, the combination of better chromosomes will produce better generations. Among chromosomes, mutations sometimes occur which will lead to improved future generations. Based on this concept, the genetic algorithm attempts to problem solve. The genetic algorithm makes use of three basic genetic operators: selection, crossover, and mutation.

4.1. Problem Coding

To solve a problem using a genetic algorithm, a model should be proposed to code the problem into a string. For task scheduling, each task is considered as a gene and the combination of these genes indicates the chromosome. Note that each one of these chromosomes will be a proposed model in task scheduling. As ***Figure 3*** illustrates, each gene consists of a number that identifies a processor.

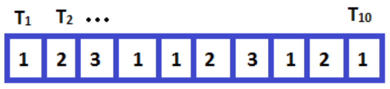


Figure 3. A chromosome for the task scheduling problem

Figure 3 shows one sample of chromosome for a graph in ***Figure 1***, meaning that task numbers 1, 2, 3 and 4 are run on CPU numbers 1, 2, 3 and 1 respectively. This would be a proposed model for task scheduling.

4.2. Selection Opeator

The rank-based wheel selection mechanism is used in EGA, as done in [15]. First, the chromosomes are sorted based on their qualities. The position of chromosomes in the list are defined as a chromosome rank and is shown as $R_i, i = 1,2,...,\varepsilon$, in which ε is the number of chromosomes in the population of the genetic algorithm. The current study employs a ranking in which $R_x = \varepsilon$ is assigned to the best chromosome of X and $R_y = \varepsilon-1$ to the second best chromosome of Y and so on. A fitness value corresponding to each chromosome is defined in ***Eq. (1)***:

$$f_i = \frac{2.R_i}{\varepsilon.(\varepsilon+1)} \tag{1}$$

Note that the amount of 0 and 1 are normalized based on the position of chromosomes in the ranking list. It is important that the rank-based selection mechanism is regarded as a static model since the probability of survival (f_i) is not dependent on generation but on the position of chromosomes on the list. For example, consider a sample of Five chromosomes in which chromosome 1 is the best quality (R_1=5), chromosome 2 is the best chromosome after chromosome 1 (R_2=4) and so on. In this case, the fitness value assigned to chromosomes is {0.33,0.26,0.2,0.13,0.06} and the associated intervals for the roulette wheel are {0-0.33,0.34-0.6,0.61-0.8,0.81-0.93,0.94-1} [15].

The EGA process applies parent selection for crossover and mutation by using the mentioned selection mechanism. This process is performed by replacement, meaning that one assumed chromosome can be selected several times as one parent, but the members in the crossover operator are different from each other.

4.3. Crossover Operator

Crossover operator is a powerful step in the genetic algorithm. In this algorithm, the operator randomly selects the position of gene and replaces the sequence before and after that gene in two chromosomes, thus producing two offsprings [16]. For example, the string 1 2 3 2 1 3 2 2 3 1 and 3 3 2 2 3 2 1 3 1 1 can be mixed after the randomized selection of the third gene and two offsprings 1 2 3 2 1 3 2 3 1 1 and 3 3 2 2 3 2 1 2 3 1, are composed. The aim of this operator is to stimulate inherited genes and the guaranteed population change. The crossover probability should be high in the early steps and average in the final steps so that the search space is properly explored. Hence, the probability crossover is defined as ***Eq. (2)***:

$$P_c(n) = P_c + \frac{n}{G}.\alpha \tag{2}$$

where P_c (n) is the probability of crossover in generation (n), G is the illustrator of the number of generations, P_c is the initial probability of crossover, and α is the coefficient between (-1,-P_c).

4.4. Mutation operator

The mutation operator produces changes in each chromosome with a low probability. This operator was found to search the new region in the search space and is used to escape the local optimum, where the algorithm is close to convergence [15]. The mutation is performed on each chromosome in an attempt to make the selected chromosome better than before. In the EGA, the mutation operator is described as follows:

For the mutation action of the present paper, a randomized amount between 0 and 1 is selected for each gene and then the randomized number for each gene is assessed as to whether it is lower than the amount of probability of mutation or not. Then the gene is omitted and transmitted to the end of the string. Similar to the crossover operator, the possibility of mutation that is lower in the first generation and increases by the final generation. Therefore, the probability of mutation is defined as ***Eq. (3)***.

$$P_m(n) = P_m + \frac{n}{G}.\alpha \tag{3}$$

where P_m(n) is the possibility of mutation in generation n, P_m is the initial probability of mutation, and α is the coefficient between (0, P_m).

4.5. Forced Reserved Strategy

In a study by Li et al. [17] forced reserved strategy was employed to maintain solutions of high quality for future generations. The current paper utilizes the same strategy. this is a type of reservation that can guarantee finding the optimized solution as soon as possible. To do so, 20% of the previous top generations are directly transmitted to the next generation without any modification. Assume that two parent individuals are selected and two new offspring individuals are created. Then, the mutation operator is utilized on the two offspring individuals and two chromosomes are created. The fitness of each of the four chromosomes is analyzed and the two top chromosomes among these four are transferred to the next generation.

4.6. Fitness Function

Fitness function is a special case of the objective function used to determine the extent to which chromosomes are worthy or not. As mentioned earlier, each chromosome can be a signal for finding the scheduling solution. Therefore, these solutions can be assessed based on each other and based on those which are advantageous to other chromosomes. From this, a proposed model of the genetic algorithm can be designed. This function is actually responsible for this evaluation and assigns a real value to each chromosome. The fitness function in EGA is defined as ***Eq. (4)***:

$$F_i = \frac{1}{M_i} \tag{4}$$

where M_i, is the make span related to chromosome *i*. Note that the focus is on finding a scheduling model that has a minimum make span, which shows that a higher quantity assigned numbers to a chromosome indicates the chromosome's higher priority. To achieve a make span, the start and finish time of each task in each processor should be identified and the finish time of the final task is regarded as a make span. In current study the start and finish time for each task is determined by as ***Eqs. (5)*** and ***(6)***:

$$ST(t_i) = MAX\{MAX\{ET(DT_K + C(DT_K)\}, FTP_j(t_i)\} \tag{5}$$

$$ET(t_i) = ST(t_i) + RT_{P_j(t_i)} \tag{6}$$

in which $ST(t_i)$ is indicative of the start time of task *i*, DT_k is the tasks prior to task *i*, C is the communication cost between the prior task and task *i*, and FTP_j (t_i) is the release time of processor *i* to which task *i* is assigned. In addition to $ET(t_i)$ represents the finish time of task *i* and RT is the running time of task *i* on processor *j*.

4.7. Final Scheduler

In the present work's suggested method, EGA is applied to create several proposed models for task scheduling. At first, the use of EGA on tasks and processors leads to several scheduling models. In fact, EGA takes into account the priority of tasks and attempts to produce a minimum make span. The task running order in each processor is considered as a default order based on the number of each task. It might be possible that, in each processor, the make span changes in terms of the changing tasks order and decreases in some cases. For this reason, after EGA is run and some proposed model are obtained, the Earliest Finished Time (EFT) for each task in each processor is applied by Algorithm 1.

Algorithm 1:

```
Input: proposed scheduling by genetic algorithm
Output: produce a final scheduler
1: for each processor $P_i$
2:     for each task $t_i$ that assigned to $P_i$
3:     get minimum $ET(t_i)$ and put it in the queue
       that belonging to $P_i$
4:    end
5: end
```

In fact, after receiving the proposed EGA models, the order of task execution for each processor is identified. In this way, the task in each processor that finishes sooner than other tasks are sent to be executed first.

4.8. The Analysis of The Convergence of The Algorithm

To theoretically prove the convergence of the algorithm, Markov's chain is used, which reveals that this algorithm can be turned into a homogeneous Markov model. The Markov process is a stochastic process in which the random variable value in sample n is only related to sample n-1 [18]. If there is a discrete space for a Markov chain, that process is named Markov's chain [18], in which time is assumed to be discrete and is defined as ***Eq. (7)***.

$$P\{X_{k+1} = x_{k+1}|X_k = x_k, \dots, X_0 = x_0\} = P\{X_{k+1} = x_{k+1}|X_k = x_k\} \tag{7}$$

where X_k presents the state value in step k. If the transition probability from one state to another is independent of time, the Markov chain is named as a homogenous Markov chain [18].

Hence, for modeling EGA to Markov chain, assume G(n) is the generation (n) in EGA and is known as system states and f_n is the maximum fitness of chromosomes in generation n. Since the number of generations is countable and discrete, the Markov chain state shows a generation in EGA. As mentioned earlier, to produce a new generation, the current generation is utilized, as defined in ***Eq. (8)***.

$$P\{G(n+1) = f_{n+1}|G(n) = f_n, \dots, G(0) = f_0\} = P\{G(n+1) = f_{n+1}|G(n) = f_n \tag{8}$$

Since the production of a new generation is dependent on the current generation, crossover, mutation, and selection operations and is also independent of time, it is concluded that this algorithm is a homogenous Markov chain (based on the definition of the homogenous Markov's chain). In Consideration that 20 percent of the EGA population is sent to the next generation, the system remains in the same state or moves on to the next state; this signifies that it can never return from the current state to its previous one. In fact, probability P_{ij} is equal to zero as P_{ij} is the probability of going from state *i* to state *j* if $j<i$. Therefore, it can be said that EGA settles to a steady state after several steps, and it converges to a global optimal solution.

5. Experiments

The proposed method is programmed by MATLAB, the R2009a version, and executed on Genuine Intel(R) processor CPU U7300 @ 1.30GHz. In EGA, the initial population is considered 50 and the maximum evolution generation is 80, as specified in ***Figures 5, 6*** and ***7***.

In this section, the performance of the proposed method is investigated in case studies mentioned in the previous algorithm [3,7,4,13,19]. In [3] and [2], the algorithm is executed on a directed acyclic graph (***Figure 1***). The results gathered from the proposed algorithm and the previous one are found in Table 1. The results assessed on their make span and indicate that EGA is no worse than other algorithms while also better than others in some cases.

Table 1. COMPARISON OF PREVIOUS ALGORITHM AND EGA RESULTS

Number of Tasks	Obtained Make Span using EGA	Obtained Make Span of Previous Algorithms	Case Study of Previous Papers
9	15	23	[13]
8	66	66	[1]
10	73	73	[4]
10	117	122	[2]
10	117	118	[3]

As shown in ***Table 1***, the EGA make span, compared to that of the algorithm presented in [13], improved by 7 time units, by 5 time units compared to the algorithm in [2], and by 1 time unit compared to the algorithm in [3]. However, it did not improve in comparison to the algorithms in [1] and [4]. This may be due to the specific examples presented in these papers.

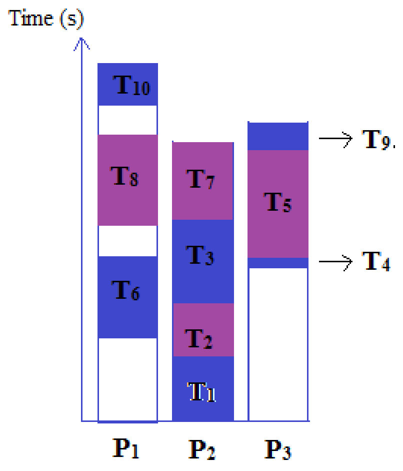


Figure 4. Problem schedule using the proposed algorithm for DAG from ***Figure 1***

Figure 4 presents the scheduling by EGA for the example in ***Figure 1***. As shown tasks 1, 2, 3, and 7 are assigned to processor 2 whose start times are 0, 21, 39, and 66 and finish times are 21, 39, 66, and 99 respectively. Tasks 4, 5, and 9 are assigned to processor 3 with start times of 50, 54, and 89 and finish times of 54, 89, and 97 respectively. Tasks 8, 6, and 10 are assigned to processor 1 and their start times are 28, 68, and 104 and finish times are 54, 94, and 117 respectively.

Figure 5 illustrates the fitness curve for EGA in three runs from the case study of [13]. Based on the graph, the algorithm in the first run has the best fitness with a 15 make span; the solution is achieved in generation 13 without any modifications. The convergence of EGA is clear in the graph. In the second and third runs the algorithm produces a make span of 16 indicating that the solution occurs in generation 38 and 5 respectively.

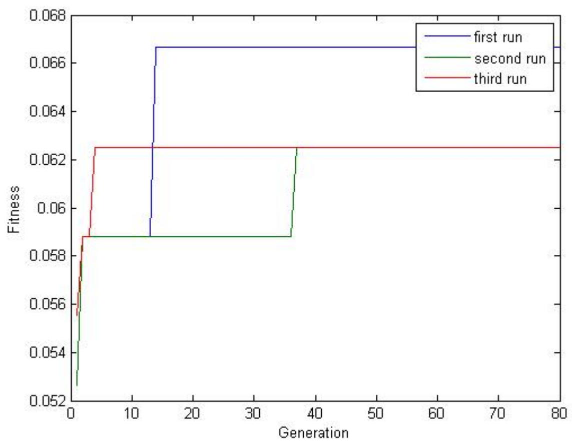


Figure 5. Fitness curve using EGA for the case study of [13]

Figure 6 shows the EGA fitness curve in three runs from the case study of [1]. On the basis of the graph, it is obvious that the algorithm achieves the best fitness in the second execution with a 72 make span; the solution is reached in generation 8 with no changes. The convergence is clear in the graph. In the first execution the algorithm shows a make span of 76 in generation 5.

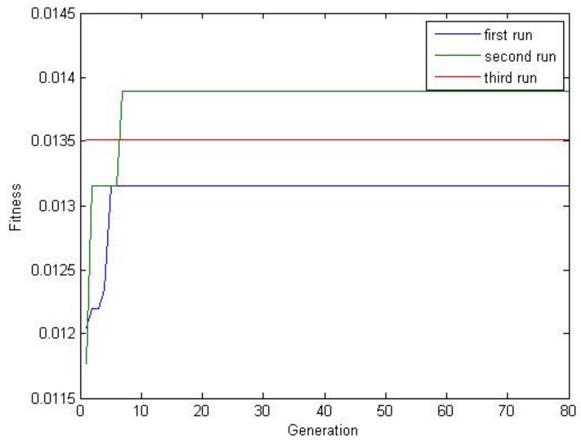


Figure 6. Fitness curve using EGA for the case study of [1]

In the third execution the make span is 74 with the solution obtained in the same first randomized population. Notice that this curve indicates the evolution of the genetic algorithm. In fact, it is the achieved make spans of the proposed genetic algorithm models which show that the order of task executions in processors is based on the default number of tasks. Therefore, the achieved make spans do not deliver the best solution. However, if algorithm 1 is applied to a model with a make span of 76 and the order of tasks is identified based on that, then the make span of 66 is the solution.

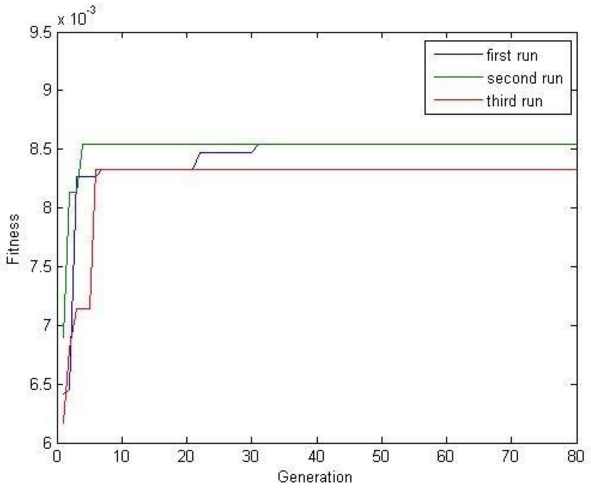


Figure 7. Fitness curve for DAG from ***Figure 1***

Figure 7 shows the fitness curve of EGA in three runs, which is used in each execution of a proposed problem in ***Figure 1***. Based on the graph, the algorithm achieves its best fitness in the first and second execution, with a make span of 117. According to the graph, in its second performance, the algorithm is faster in finding a solution than in its first execution, as the solution is achieved in generation 5 in the first execution and generation 31 in the next one. In the third execution, the algorithm produces a make span of 120 in generation 7.

To analyze the effectiveness of EGA, it is executed many times, for example, in ***Figure 1*** and the results are shown in ***Table 2***. In the first nine replications of the tests, the population is 50 and the number of generations is 80. In seven out of the nine tests, the make span is 117 and two others reach 118, indicating that the algorithm works well with make spans close to each other in successive replications. In the next six replications, the population decreases to 20. The make spans are 117 in two cases and the worst make span among all of those six executions is 129. In the next five replications, the population increases from 20 to 30 and the number of generations decreases from 80 to 20. In these tests, the make spans are 117 in 3 out 5 times and the worst make span is 121. According to these results, it can be concluded that the algorithm achieves the solution in the first generations and is dependent on the first population rather than on the number of generations. It is the forced reserved strategy that leads to the solution in the first generations.

Table 2. RESULTS OF 20 ITERATIONS OF THE PROPOSED ALGORITHM FOR DAG FROM ***FIGURE 1***

Iterations	Number of the Initial Population	Number of Generations	Make Span	Proposed Scheduling
1	50	80	117	2 2 2 3 3 1 2 1 3 1
2	50	80	117	2 2 2 3 3 1 2 1 3 1
3	50	80	117	2 2 2 3 3 1 2 1 3 1
4	50	80	117	2 2 2 3 3 1 2 1 3 1
5	50	80	118	2 1 2 3 3 2 2 1 3 1
6	50	80	118	2 1 2 3 3 2 2 1 3 1
7	50	80	117	1 1 1 2 3 2 1 2 3 2
8	50	80	117	1 1 1 2 3 2 1 2 3 2
9	50	80	117	1 1 1 2 3 2 1 2 3 2
10	20	80	129	1 2 1 1 3 1 3 1 3 1
11	20	80	121	2 1 2 2 3 2 1 2 3 2
12	20	80	117	2 2 2 3 3 1 2 1 3 1
13	20	80	124	2 3 2 2 1 3 2 2 1 2
14	20	80	117	1 1 1 2 3 2 1 2 3 2
15	20	80	121	2 1 2 2 3 2 1 2 3 2
16	30	20	117	2 2 2 3 3 1 2 1 3 1
17	30	20	118	2 1 2 1 3 2 2 1 3 1
18	30	20	117	1 1 1 2 3 2 1 2 3 2
19	30	20	117	2 2 2 3 3 1 2 1 3 1
20	30	20	121	2 1 2 2 3 2 1 2 3 2

6. Conclusion

The present study delves into solving the problem of task scheduling in heterogeneous systems based on EGA. First applying this algorithm, several proposed models for task scheduling are obtained. Then, based on using the Earliest Finished Time (EFT) on the EGA models, the order of tasks on each processor is determined and, finally, the scheduling is produced. EGA obtains the solution by utilizing forced reserved strategy faster and in the first generations. The comparisons reveal that this is more effective when compared to previous algorithms and so achieves better make spans.

References

[1] Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Information Sciences*, *270*, 255-287.

[2] Arabnejad, H., & Barbosa, J. G. (2013). List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE transactions on parallel and distributed systems*, *25*(3), 682-694.

[3] Panwar, P., Sachdeva, S., & Rana, S. (2016). A genetic algorithm based scheduling algorithm for grid computing environments. In *Proceedings of Fifth International Conference on Soft Computing for Problem Solving: SocProS 2015,1*,165-173 Springer Singapore.

[4] Coffman, E. G., & Bruno, J. L. (1976). Computer and job-shop scheduling theory. *John Wiley & Sons.*

[5] Gary, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness.

[6] Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System sciences*, *10*(3), 384-393.

[7] Zomaya, A. Y., Ward, C., & Macey, B. (1999). Genetic scheduling for parallel processor systems: comparative studies and performance issues. *IEEE Transactions on Parallel and Distributed systems*, 10(8), 795-812.

[8] Kwok, Y.K., Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. (CSUR) 31(4).

[9] Topcuoglu, H., Hariri, S., & Wu, M. (2002). Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel and Distributed Systems*, 13(3), 260-274.

[10] Canon, L.C., Jeannot, E., Sakellariou, R., & Zheng, W. (2008), Comparative Evaluation of the Robustness of Dag Scheduling Heuristics. *Grid Computing: Achievements and Prospects, S.Gorlatch, P. Fragopoulou, and T. Priol, eds., Springer,* 73-84.

[11] Boeres, C., Filho, J.V., & Rebello, V. E. F. (2004). A Cluster-Based Strategy for Scheduling Task on Heterogeneous Processors. *Proc. 16th Symp. Computer Architecture and High Performance Computing,* 214-221.

[12] Cirou, B., & Jeannot, E. (2001). Triplet: A Clustering Scheduling Algorithm for Heterogeneous Systems. *Proc. Int'l Conf. Parallel Processing Workshops*, 231-236.

[13] Jiang, Y. S., & Chen, W. M. (2015). Task scheduling for grid computing systems using a genetic algorithm. *The Journal of Supercomputing*, *71*(4), 1357-1377.

[14] Holland, J. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press, Ann Arbor, MI, USA.*

[15] Agustı, L. E., Salcedo-Sanz, S., Jiménez-Fernández, S., Carro-Calvo, L., Del Ser, J., & Portilla-Figueras, J. A. (2012). A new grouping genetic algorithm for clustering problems. *Expert Systems with Applications*, *39*(10), 9695-9703.

[16] M. Mitchell, (1999). An Introduction to Genetic Algorithms. Cambridge, Massachusetts London, England: Massachusetts Institute of Technolog.

[17] Li, F., Da Xu, L., Jin, C., & Wang, H. (2012). Random assignment method based on genetic algorithms and its application in resource allocation. *Expert Systems with Applications*, *39*(15), 12213-12219.

[18] Gebali, F. (2015). Analysis of Computer Networks. *Cham Heidelberg New York Dordrecht London: Springer.*

[19] Panwar, P., Lal, A.K., & Singh, J. (2012). A Genetic algorithm based technique for efficient scheduling of tasks on multiprocessor system. *In: Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011),* 911–919. *Springer India.*