International Journal of Cloud Applications and Computing
Volume 7 • Issue 4 • October-December 2017

An Energy-Aware Task Scheduling in the Cloud Computing Using a Hybrid Cultural and Ant Colony Optimization Algorithm

Poopak Azad, Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran
Nima Jafari Navimipour, Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

ABSTRACT

In a cloud environment, computing resources are available to users, and they pay only for the used resources. Task scheduling is considered as the most important issue in cloud computing which affects time and energy consumption. Task scheduling algorithms may use different procedures to distribute precedence to subtasks which produce different makespan in a heterogeneous computing system. Also, energy consumption can be different for each resource that is assigned to a task. Many heuristic algorithms have been proposed to solve task scheduling as an NP-hard problem. Most of these studies have been used to minimize the makespan. Both makespan and energy consumption are considered in this paper and a task scheduling method using a combination of cultural and ant colony optimization algorithm is presented in order to optimize these purposes. The basic idea of the proposed method is to use the advantages of both algorithms while avoiding the disadvantages. The experimental results using C# language in cloud azure environment show that the proposed algorithm outperforms previous algorithms in terms of energy consumption and makespan.

KEYWORDS

Ant Colony Optimization Algorithm, Cultural Algorithm, DAG, Task Scheduling

1. INTRODUCTION

Cloud computing is a popular phenomenon (Chiregi & Navimipour, 2016; Sheikholeslami & Navimipour, 2017) in which shared resources are prepared to end-users in an on-demand fashion that brings many advantages, including data ubiquity, the flexibility of access, high availability of resources, and scalability (Bouarara, Hamou, Rahmani, & Amine, 2014; Kumar, Ashok, & Subramanian, 2012). Cloud computing focuses on commercial resource provision and allows customers to use the computing resources presented by multiple service providers (Kim & Jo, 2016; Mohammadi, 2017). It is a model of service delivery and access mechanism where virtualized resources are provided as a service over the Internet (Milani & Navimipour, 2016; Anil Singh, Dutta, & Singh, 2014; Sood, 2013). It follows a pay-per-use model and can be dynamically reconfigured to satisfy user requests via on-the-fly virtual resources (Navimipour, Rahmani, Navin, & Hosseinzadeh, 2015; Vakili & Navimipour, 2017).

DOI: 10.4018/IJCAC.2017100102

The main problems of cloud computing are dynamism, requiring continuous monitoring of requests and resources, handling of ever changing requirements, schedules and prices, selecting appropriate services and plans to meet overall objectives of the cloud (Aznoli & Navimipour, 2017; Chowhan, Shirwaikar, & Kumar, 2016). Cloud computing presents many services to users (Alami Milani & Jafari Navimipour, 2016; Ezugwu, Buhari, & Junaidu, 2013) such as Software as a Service (SaaS) (Alkhanak, Lee, & Khan, 2015), Infrastructure as a Service (IaaS) (Mehta & Gupta, 2013), Platform as a Service (PaaS) (Shao, Wang, & Mei, 2012) and Expert as a service (EaaS) (Navimipour et al., 2015). These services can then be accessed through a cloud client which could be a web browser, mobile app, and so on (Chong, Wong, & Wang, 2014).

On the other hand, task scheduling on distributed computing environments such as cloud computing is an interesting issue (Keshanchi, Souri, & Navimipour, 2017). In order to arrange the performance of the task in a cloud, an efficient task scheduler is requested in which applications should divide into subtasks (Navimipour, 2015a, 2015b). These subtasks are shown as a directed acyclic graph (DAG). The type of task scheduling greatly affects the energy consumption of a cloud datacenter, if the task is not properly scheduled can increase energy consumption, therefore an energy aware task scheduling can save lots of energy. In the cloud computing environment, various types of users perform their tasks (Aarti Singh & Malhotra, 2015). Each of them has entirely different resource requirements (Habibi & Navimipour, 2016).

By considering the important role of task scheduling in cloud computing, this paper is aimed to propose an efficient task scheduling algorithm including two important criteria, the makespan, and energy consumption. During the last few years, the high price of energy consumption has become a critical issue (Koomey, 2011) and cloud providers faced with the pressure of minimizing their energy consumption as well as their amount of CO2 emissions. Since ant colony optimization (ACO) algorithm has long iteration time and convergence time is uncertain. On the other hand, the cultural algorithm has been successfully applied to optimization problems and has advantages in overcoming some weaknesses of conventional optimization methods (Yang & Gu, 2014), the ACO algorithm is combined with a cultural algorithm to improve the performance of the task scheduling algorithm and deal with that issues. Briefly, the main purpose of this paper is proposing a new hybrid algorithm using the cultural algorithm and ACO algorithm for minimizing makespan and energy consumption.

The rest of this paper is organized as follows. Related mechanisms are reviewed in Section 2. An introduction to ACO algorithm, cultural algorithm, and the proposed algorithm are discussed in Section 3. Empirical experiments of evaluating the improved algorithm are conducted in Section 4. Finally, the paper concludes in the last section.

2. RELATED WORK

Since task scheduling is an NP-hard problem, many researchers presented the nature-inspired optimization algorithms for task scheduling in cloud environments. In this section, some state-of-the-art mechanisms are discussed and analyzed.

Zuo (2015) has proposed a multi-objective optimization method for solving the task scheduling problem in cloud computing. They proposed a resource cost model that defines the demand of tasks on resources with more details. This model reflects the relationship between the user's resource costs and the budget costs. A multi-objective optimization scheduling method has been proposed based on the proposed resource cost model. This method considers the makespan and the user's budget costs as limitations of the optimization problem, to achieve multi-objective optimization of both performance and cost. An improved ACO algorithm has been proposed to solve this problem. It reduces the makespan, prevented from falling into a locally optimal solution and optimized the user costs.

Also, Xu (2014) has proposed a task scheduling scheme on heterogeneous computing systems using multiple priority queues based on genetic algorithm. The basic idea of this approach is to exploit the advantages of both evolutionary-based and heuristic-based algorithms while avoiding their disadvantages. The proposed algorithm incorporates a genetic algorithm to assign a priority to each subtask while using a heuristic-based earliest finish time (EFT) approach to search a solution for the task-to-processor mapping. The proposed method also employs crossover, mutation, and fitness function. The advantages of the algorithm are the low makespan and high efficiency but it suffers from slow convergence and high complexity.

On the other hand, the growth of energy consumption has been explosive in current data centers, supercomputers, and public cloud systems. This explosion leads to the greater defense of green computing, and many efforts and works are focused on the task scheduling in order to reduce dissipation of energy. Tang (2016) has proposed a dynamic voltage and frequency (DVFS)-enabled energy efficient workflow task scheduling algorithm. The proposed method combines the relatively inefficient processors by reclaiming the slack time, to calculate the initial scheduling order of all tasks, and obtains the whole makespan and deadline based on heterogeneous EFT (HEFT) algorithm. Finally, the tasks can be distributed in the idle slots under a lower voltage and frequency using DVFS technique. The obtained results showed that the total power consumption for various parallel applications is improved. But the algorithm does not consider overheads and parameters in the actual presence of a heterogeneous environment.

Furthermore, Khan (2012) has suggested an approach called constrained EFT (CEFT) to provide better schedules for heterogeneous systems using the concept of the constrained critical paths. In contrast to other approaches used for heterogeneous systems, the CEFT strategy includes a broader view of the input task graph. Furthermore, the statically generated constrained critical paths may be efficiently scheduled in comparison with other methods. The proposed method outperforms the well-known HEFT, DLS, and LMT strategies by producing shorter schedules for a diverse collection of task graphs. However, this method produces unstable results in large problems.

Keshanchi and Jafari Navimipour (2016) have proposed a new task scheduling algorithm in a cloud environment using multiple priority queues and a memetic algorithm. The method uses a genetic algorithm along with hill climbing to assign a priority to each subtask while using a heuristic-based EFT approach to search a solution for the task-to-processor mapping. The basic idea of the approach is using the advantage of the memetic algorithm to increase the convergence speed of the solutions. The simulation results show that the algorithm outperforms the makespans of the three heuristic algorithms, but it has high complexity and running time.

Mentioned works have some advantages and disadvantages that listed in Table 1. Due to the problems of previous works in this paper, we introduce a new hybrid energy-aware approach that detailed in Section 3.

3. PROPOSED METHOD

A task scheduling algorithm in a cloud environment is proposed in this section to exploit the advantages of both ACO and cultural algorithms. The system model is presented in Section 3.1 and the details of our task scheduling model are presented in Section 3.2.

3.1. System Model

In this work, the cloud application model has a set of P heterogeneous processors that are fully interconnected with a high-speed network. The inter-processor communications are performed at the same speed on all links to simplify task scheduling model same as the model in (Dorigo & Gambardella, 1997). In this system, each subtask can be run only on one processor, and all of them must be scheduled.

Table 1. Task scheduling comparison

Technique	Approach	Advantages	Disadvantages
Zuo (2015)	A multi-objective optimization method for task scheduling	● High performance ● Preventing from falling into a locally optimal solution ● Low makespan Optimizing user costs	● The performance do not compare with existing algorithms
Xu (2014)	A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues	● Low makespan ● High efficiency	● Slow convergence ● High complexity
Tang (2016)	An energy efficient task scheduling algorithm based on DVFS	● Low power consumption ● Low makespan	● Low reliability ● Not considering overheads and parameters in the actual presence of a heterogeneous environment
Khan (2012)	Based on constrained critical paths	● Low makespan	● Producing unstable results in large problems
Keshanchi and Jafari Navimipour (2016)	Priority based task scheduling using a memetic algorithm	● Low makespan ● Fast convergence	● High complexity ● High running time

Time dependency relationship should be considered when two dependent subtasks are assigned to different processors.

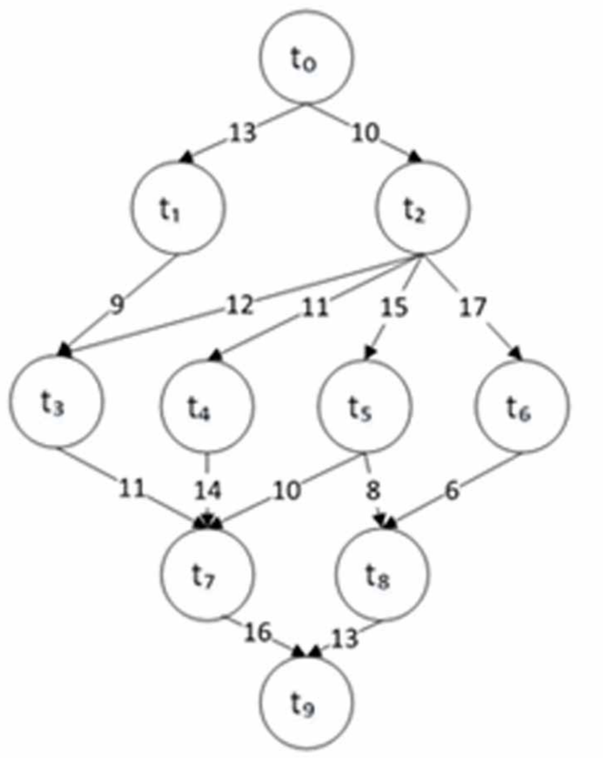
A static computational model is considered in this model and the relation and execution precedence is predetermined and it doesn't change during the task scheduling or performance. The task scheduling problem is shown by using directed acyclic graph (DAG) to represent the tasks and their dependencies. In DAG, the vertices represent tasks and the edges represent execution precedence between tasks as shown in Figure 1. The edges of the graph are labeled with communication cost. In Table 2, the first column shows the tasks of the application of Figure 1, columns 2 to 4 represent the computation cost of each task on processors and the last column shows the average computation cost of each task on all processors.

3.2. Hybrid Algorithm

The ACO methodology was originally introduced by Dorigo (1997). The general principal of ACO is that the pheromone information reflects the outcomes of the decision which have been made by former ants to find good solutions (Kang, Zhang, Lin, & Lu, 2014). It is inspired by the observation of the behavior of ants when trying to reach a food source (Tumeo, Pilato, Ferrandi, Sciuto, & Lanzi, 2008). At first, the ants wander randomly. When an ant finds a food source, it walks back to the colony leaving pheromones that show the path of the food (Navimipour & Milani, 2016). When other ants come across the pheromones, they are likely to follow the path with their own pheromones as they bring the food back. As more ants find the path, it gets stronger until there are a couple streams of ants traveling to various food sources near the colony. By dropping pheromones every time, they bring food. Shorter paths are more likely to be stronger, hence the solution is optimized. The pheromone also has evaporation factor. Over time, the pheromone trail starts to evaporate and less reinforced routes will slowly disappear (Tumeo et al., 2008).

In ACO algorithm, convergence is guaranteed, but convergence time is uncertain. The traditional ACO algorithm is prone to stagnation. In order to avoid the disadvantages of ACO

Figure 1. A simple DAG with 10 nodes and 14 edges



algorithm, the ACO is combined with a cultural algorithm for solving the task scheduling problem, which could essentially improve the performances of the algorithm and deal with long iteration time and local convergence issues. Both algorithms are based on population and they share information among the population. ACO algorithm gets benefits from the dual inheritance of cultural algorithm.

The Cultural algorithm is a branch of evolutionary computation that has two components: the belief space which is the evolutionary process from the experiences and knowledge acquired; and the population space which is a group of individuals from the specific space (Reynolds & Zhu, 2001). These two components communicate with each other through a supporting communication protocol (Zhu & Wang, 2016). The dual inheritance mechanism makes the cultural algorithm a self-adaptation system that enables global evolutionary information to be more fully utilized. The belief space is updated after each iteration by the best individuals using a fitness function that determines the performance of each individual in the population. Figure 2 represents the framework of the cultural algorithm.

Acceptance and influence are considered as the major operations of the cultural algorithm. Other operations are performed inside belief or population space independently. Therefore, it is possible to set other algorithms into cultural algorithm framework by adding specific logics into belief and population spaces and performing acceptance and influence operations between them (Zhu & Wang, 2016). In the proposed method, ACO is used to processor selection and the cultural algorithm is used to select the best solution and avoid falling into local optimum solution using its operators. The steps of the method are shown in Figure 3.

Table 2. Computation cost of DAG in Figure 1

Task	p_0	p_1	p_2	$\bar{\omega}$
t_0	9	11	7	9
t_1	8	7	6	7
t_2	12	9	9	10
t_3	15	13	17	15
t_4	9	7	11	9
t_5	6	10	8	8
t_6	16	17	15	16
t_7	13	8	12	11
t_8	12	18	15	15
t_9	8	7	6	7

Figure 2. Framework of cultural algorithm (Reynolds & Zhu, 2001)

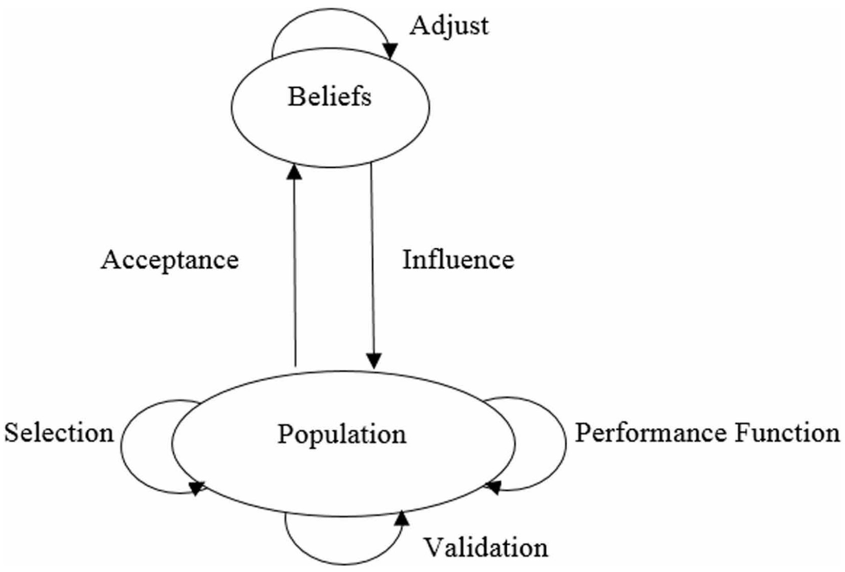
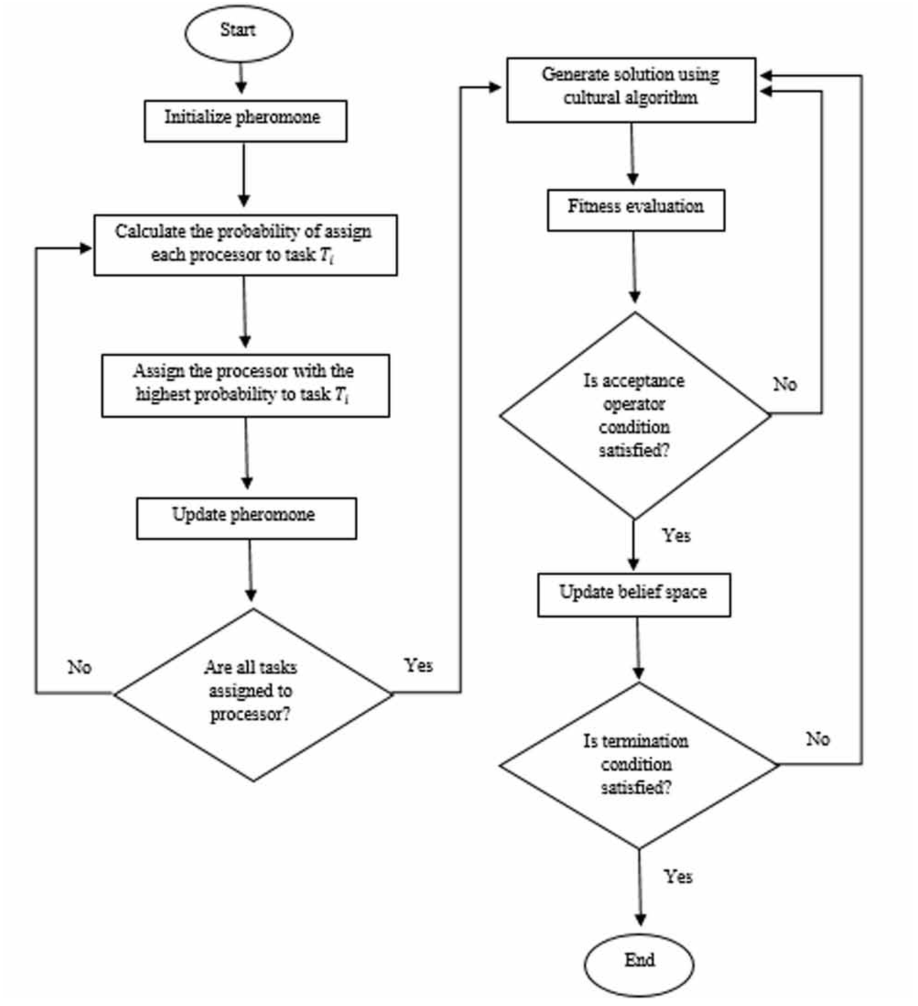


Figure 3. Flowchart of the proposed method


Start
Initialize pheromone
Calculate the probability of assign each processor to task T_i
Assign the processor with the highest probability to task T_i
Update pheromone
Are all tasks assigned to processor?
No
Yes
Generate solution using cultural algorithm
Fitness evaluation
Is acceptance operator condition satisfied?
No
Yes
Update belief space
Is termination condition satisfied?
No
Yes
End

3.3. Processor Selection

In the proposed algorithm, ants select the processor based on its pheromone information and heuristic information. Every ant chooses a processor for the first task randomly. Ant k selects a task in sequence and assigns task i to processor P according to the probability information:

$$pt_i = \frac{\left[phr\left(i,j\right)\right]\left[heu\left(i,j\right)\right]^{\beta}}{\sum_{j=1}^{n}\left[phr\left(i,j\right)\right]\left[heu\left(i,j\right)\right]^{\beta}} \tag{1}$$

where $phr\left(i,j\right)$ is the pheromones information and the last completion time for the processor P_j. $heu\left(i,j\right)$ is the heuristic information and the earliest start time for task T_i on the processor P_j. Heuristic information is obtained from the equations of Section 3.2.2 that gives EFT of task T_i on processor P_k. A task can only be selected when it's all prior tasks are scheduled (Ashouraie & Jafari Navimipour, 2015). An important issue is the definition of trail pheromones for the particular processor assignment of each task. When an artificial ant has been assigned to a processor, the pheromone information is updated by applying the pheromone update rule as follows:

$$phr(i,j) = (1-\rho)\, phr(i,j) + \rho, fitness \tag{2}$$

In this equation, ρ is the pheromone evaporation rate which is between 0 and 1, and $fitness$ that is calculated using formulas in Section 3.2.3, is the fitness value. The effect of the pheromone update rule is to make the choosing of putting T_i on the processor P_j less desirable for other ants to achieve diversification because ants tend to converge to a common path. The purpose of the pheromone update rule is to encourage the ants to search for processors with less energy consumption and execution time.

3.4. Heuristic Information

Heuristic information in the processor assigning probability equation is achieved from heuristic based HEFT algorithm. This algorithm is proposed to minimize makespan without violating precedence constraints. The HEFT algorithm selects the subtask with the highest upward rank value at each step and assigns the selected task to the processor which minimizes its earliest finish time with an insertion based approach (Xu et al., 2014). The EST of the task T_i on processor P_k is represented as $EST(T_i, P_k)$:

$$EST(T_i,P_k) = \begin{cases} 0, & if\ T_i = T_{entry} \\ \max\limits_{T_j \in Pred(T_i)} AFT(T_j, P_l), & if\ P_k = P_l \\ \max\limits_{T_j \in Pred(T_i)} \left(AFT(T_j,P_l) + C(T_j,T_i) \right), & if\ P_k \neq P_l \end{cases} \tag{3}$$

The actual start time (AST) of task T_i on processor P_k is represented as $AST(T_i, P_k)$:

$$AST(T_i,P_k) = \max\left(EST(T_i,P_k), Avail(P_k) \right) \tag{4}$$

The $Avail(P_k)$ is defined as the earliest time at which the processor P_k is ready for the task execution. The EFT of task T_i on processor P_k is represented as $EFT(T_i, P_k)$:

$$EFT(T_i,P_k) = AST(T_i,P_k) + W(T_i,P_k) \tag{5}$$

The actual finish time (AFT) of a task T_i over all processors is represented as $AFT(T_i, P_k)$, P_k is the fittest processor for the task T_i:

$$AFT\left(T_i, P_k\right) = \min_{1 \le l \le m} EFT\left(T_i, P_l\right) \tag{6}$$

3.5. Fitness Function

The fitness function can directly affect the algorithm convergence and the search for an optimal solution. In this study, generally, the task scheduling objective function is considered the minimum makespan and the lowest energy as the target. The normalization process is conducted in order to create a balance of proportion of makespan and energy consumption. The makespan is derived from the following equation:

$$makespan\left(i\right) = \max\left\{AFT\left(t_{exit}\right)\right\} \tag{7}$$

Due to the scheduling proposed model, the fitness function is calculated based on the makespan and energy consumption. The fitness function algorithm is as follows:

$$fitness\left(i\right) = \frac{1}{makespan\left(i\right) \times energyConsumption\left(i\right)} \tag{8}$$

Here, i is a practical solution in order to ensure the quality of the solution, to avoid falling into local optimum and to achieve the optimal solution as far as possible. A fitness function is used to evaluate the quality of practical solutions.

3.6. Belief Space

The major operations of the cultural algorithm are acceptance and influence (Zhu & Wang, 2016). The acceptance function determines which individuals from the current population will be used to shape the beliefs of the entire population. In acceptance function, static methods use absolute ranking based on fitness values to select the top n% individuals. But dynamic methods do not have a fixed number of individuals that adjust the belief space, Instead, the number of individuals may change from generation to generation (Engelbrecht, 2007). Alternatively, the number of individuals is determined as:

$$n\mathcal{B}\left(t\right) = \frac{n_s \gamma}{t} \tag{9}$$

where n_s is the population size, $\gamma \in \left[0,1\right]$ and t is the counter of iterations. Using this approach, the number of individuals which are used to adjust the belief space is initially large by decreasing overtime.

3.7. Cultural Algorithm Operators

Three cultural algorithms operators including selection, crossover and mutation operator are presented in this section. Rolette wheel selection is used to select the solution with the highest fitness and single point crossover operator of a genetic algorithm for crossover operation. Finally, mutation operator of the genetic algorithm is used to maintain diversity.

3.7.1. Selection Operator

Several selection operators are introduced as an important part of a genetic algorithm. Selection operator prefers to select the better solutions and better individuals with high probability of its survival and mating (Abdelhalim & El Khayat, 2016). Rolette wheel method is used for this operator. This method assumes that the probability of selection is proportional to the fitness of a solution. A solution having a higher fitness value should have a higher chance to be selected. We consider $PopSize$ number of population, each characterized by its fitness $fitness_i > 0\left(i = 1, 2, \ldots, popsize\right)$. The probability p_i of each solution to be selected can be calculated according to the probability defined by the equations:

$$p_i = \frac{fitness_i}{\sum_{j=1}^{PopSize} fitness_j} \tag{10}$$

and:

$$S_i = \sum_{j=1}^{i} p_j \tag{11}$$

where S_i is the sum of p_j from 1 to i. Consequently, a more fitted solution will be selected with higher probability and it will get more offsprings. The average will stay and the worst will die off.

3.7.2. Crossover Operator

The crossover is a genetic operator to change the programming of a chromosome. A crossover is used as a procedure of replacing some of the genes in one parent by corresponding genes of the other. In the task scheduling problem, the crossover operator is combining two valid parents, whose subtasks are ordered topologically to generate two offsprings which will also be valid. Figure 4 represents the single point crossover.

3.7.3. Mutation Operator

The mutation operator is a genetic operator to maintain diversity. This operator changes genes of a chromosome and transforms it from one generation of a population to the next generation and helps the search algorithm to avoid from falling to locally optimal solutions by preventing the population from becoming too similar to each other, or cooperates with crossover operator to achieve a better solution. Figure 5 represents the mutation operator.

Figure 4. Single point crossover operator

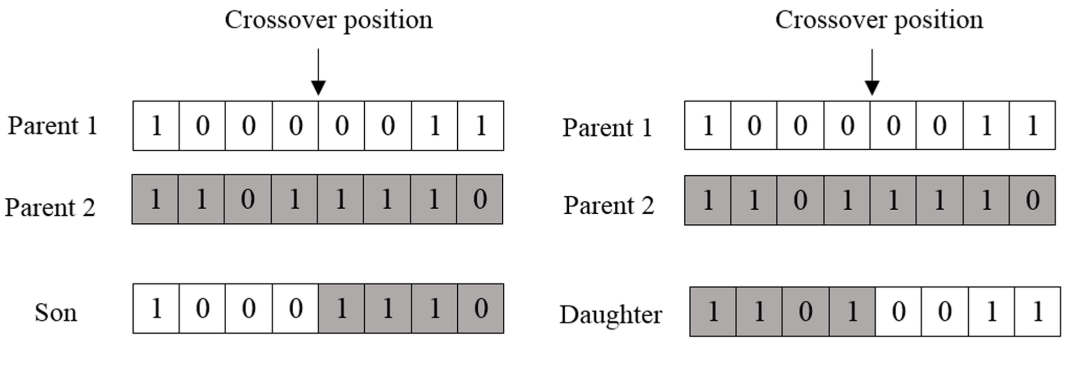
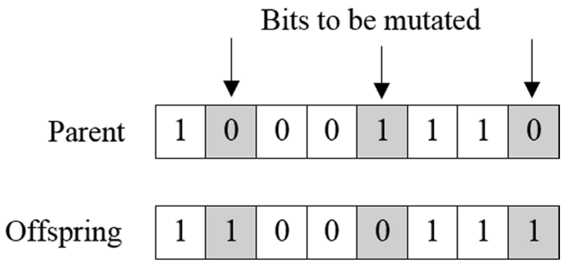


Figure 5. Mutation operator



4. EXPERIMENTAL RESULTS

This section presents simulation results to evaluate the performance of the proposed method in terms of energy consumption and makespan with ACO and HEFT-upward rank. The rest of this section is organized as follows. Simulation tools are introduced in Section 4.1. Simulation parameters and dataset are presented in Section 4.2 and 4.3, respectively. Finally, the acquired results are presented in Section 4.4.

4.1. Simulation Tools

The proposed algorithm is performed in C# programming language in cloud azure environment, similar to (Keshanchi & Navimipour, 2016; Keshanchi et al., 2017). Azure was introduced as windows azure in 2008 by Microsoft (Copeland, Soh, Puca, Manning, & Gollob, 2015) and it is a cloud computing service for building, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides SaaS, PaaS, and IaaS and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems. Simulation and algorithm running have been done on a personal computer with an Intel processor with core i5 and 33.3 GHz CPU and 4 GB RAM.

4.2. Simulation Parameters

Crossover and mutation probability are considered 0.3 and 0.7, respectively. Also, the evaporation rate of ACO algorithm is considered 0.5. Finally, energy consumption of processors is a random number between 0.001 to 0.004. These simulation parameters are shown in Table 3.

4.3. Dataset

Random generated DAGs is used in this section to evaluate the performance of proposed method compared with other methods. So the cost of computing and communication through a range of graphs randomly selected. We ran our simulation three times with 10, 20, 50 tasks and 3, 6 processor and population size in each round is considered 100, 150, 200. Table 4 represents dataset information.

Table 3. Simulation parameters

Parameters	Values
crossover probability	0.3
mutation probability	0.7
Evaporation rate	0.5
Energy consumption of processors	0.001 – 0.004
Termination condition	50 iterations

Table 4. Dataset information

Number of DAGs	50
Number of tasks	10, 20, 50
Number of processors	3, 6
Population size	100, 150, 200

4.4. Obtained Results

Figure 6 shows scheduling results of 50 graphs with 10 tasks. Total average graph completion time for the HEFT-upward rank method is 105.22, for the ACO is 102.08 and for the proposed method is 106.48. Also, the results of energy consumption associated with the results of Figure 6 are given in Figure 7 and for the HEFT-upward rank, the ACO and the proposed method, are 0.369, 0.253 and 0.204, respectively. The results indicate that the proposed method in terms of makespan is further narrowly than other methods, but it has better performance in terms of the average value of energy consumption. Figure 8 shows the convergence behavior of the solution to one of the graphs shown in Figure 6.

Figure 9 shows the makespan of 50 random graphs with 20 tasks. The total average of completion time for the HEFT-upward rank is 199.48, for the ACO is 210.55 and for the proposed method is 225. Also, the results of energy consumption associated with the results of Figure 9 are given in Figure 10 which for the HEFT-upward ranks, the ACO, and the proposed method are 0.736, 0.431, 0.282 watts, respectively. The results which are displayed in Figure 11 show the convergence behavior of the solution to one of the graphs shown in Figure 9.

Figure 12 shows the makespan of 50 random generated graph with 50 tasks. The total average of completion time for three methods including HEFT-upward rank, ACO, and proposed method are 486.78, 570 and 579, respectively. The results of energy consumption associated with the results of Figure 12 for three methods such as HEFT-upward rank, ACO, and the proposed method are1.651, 1.005, 0.756 watts, respectively. The results are plotted in Figure 13. The proposed method in terms of program completion time compared to other methods is non-optimal narrowly but quantitatively

Figure 6. Makespan of 50 random generated DAG (number of tasks = 10, number of resources = 3, number of population = 100)

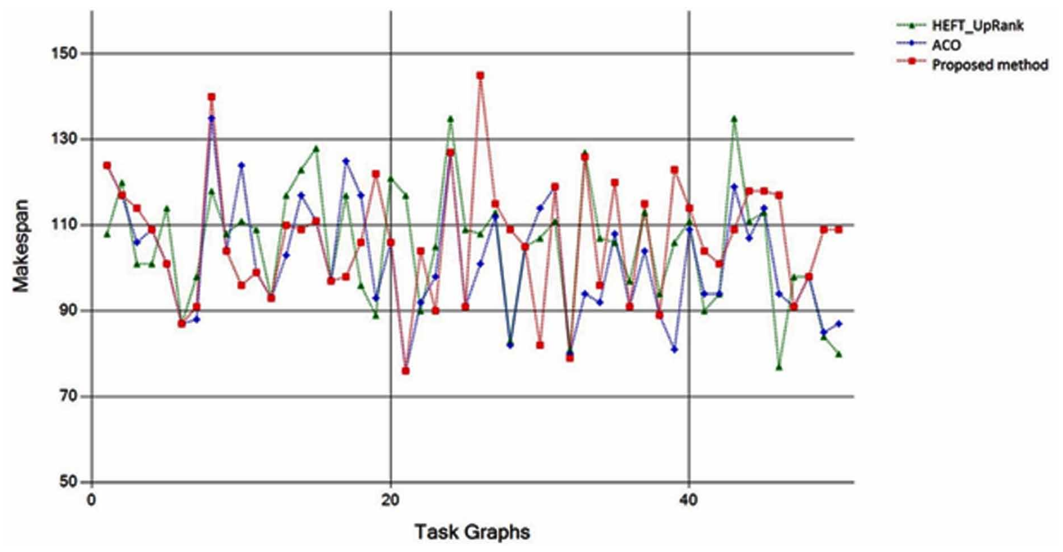


Figure 7. Energy consumption for 50 random generated DAG (number of tasks = 10, number of resources = 3, number of population = 100)

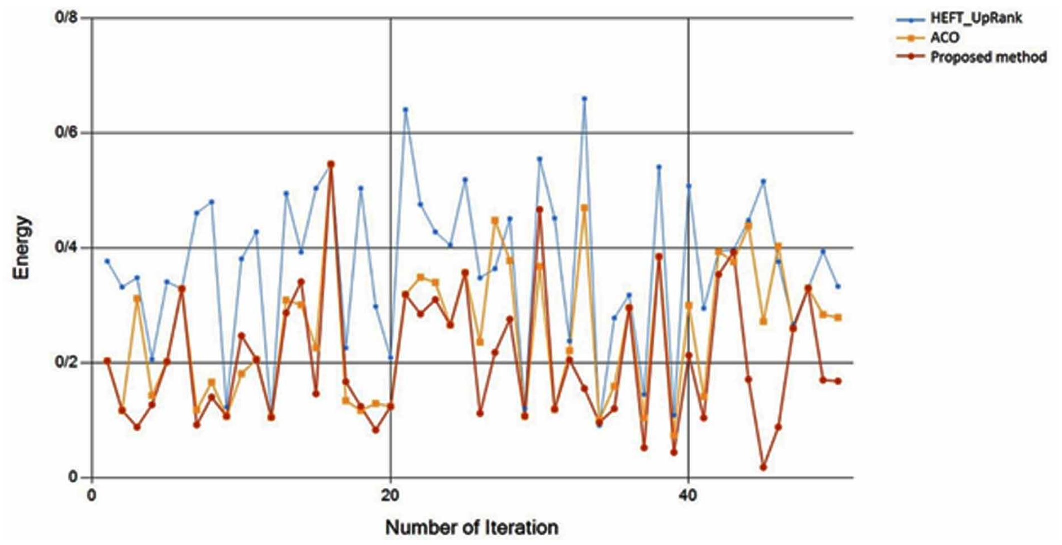
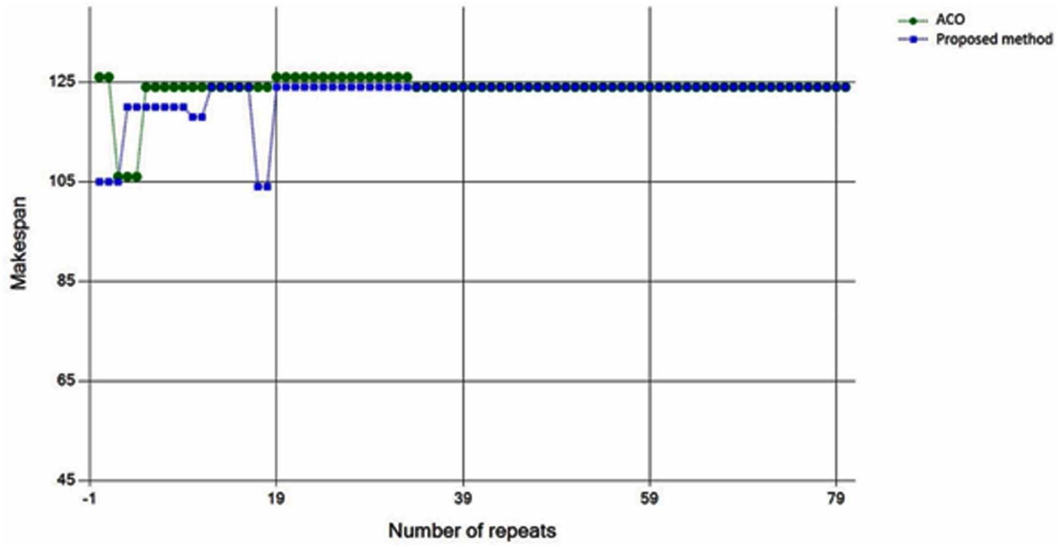


Figure 8. Convergence of the best solutions in each repeat of selected sample of 50 graphs in Figure 6



significantly minimizes the energy consumption of other methods. Figure 14 shows the convergence behavior of the solution.

4.4.1. Results for Graph of Figure 1

The Results for the graph of Figure 1 are shown in Figure 15 and 16. Completed time of three methods, including HEFT-upward rank, ACO and proposed method as a bar for graph Figure 1 with 10 tasks

Figure 9. Makespan of 50 random generated DAG (number of tasks = 20, number of resources = 6, number of population = 150)

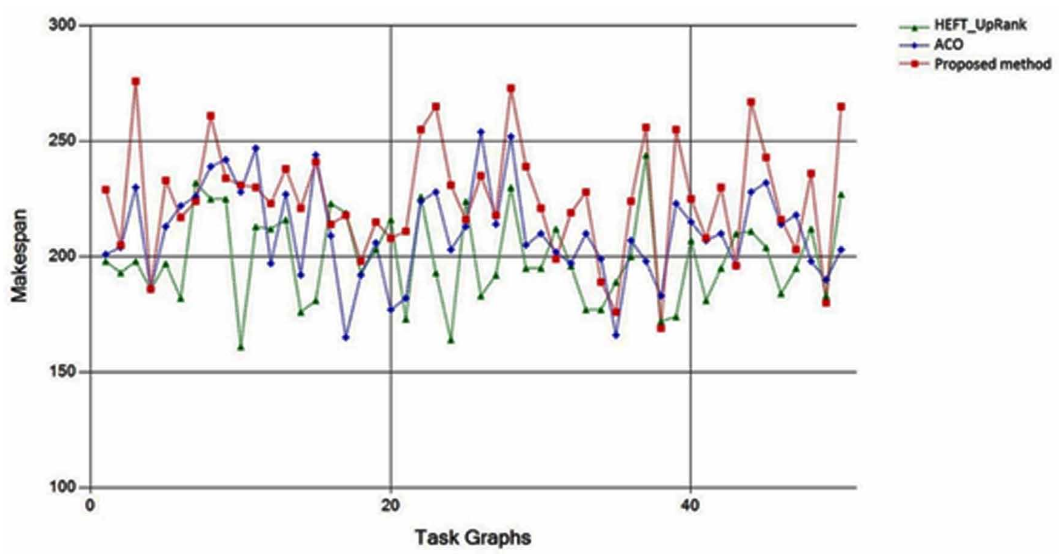
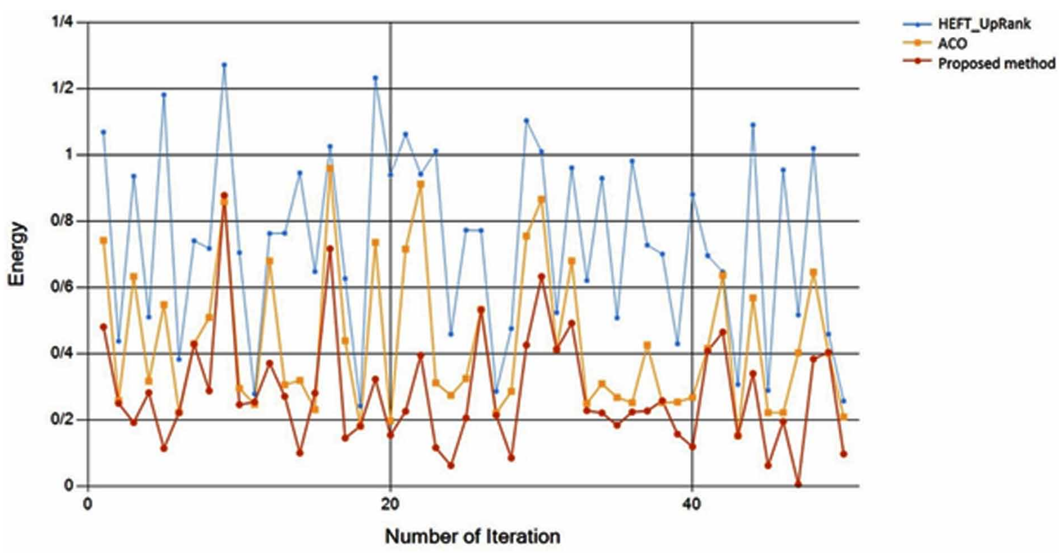


Figure 10. Energy consumption for 50 random generated DAG (number of tasks = 20, number of resources = 6, number of population = 150)



are displayed in Figure 15. The results of the test for makespan of the HEFT-upward rank is 82, the ACO is 76 and the proposed method is 77. Also, Figure 16 shows the amount of energy consumption for these three techniques. The amount of energy consumption in scheduling for HEFT-upward rank is 0.297, for ACO is 0.225 and for proposed method is 0.216 watts. The results show that the

Figure 11. Convergence of the best solutions in each repeat of selected sample of 50 graphs in Figure 9

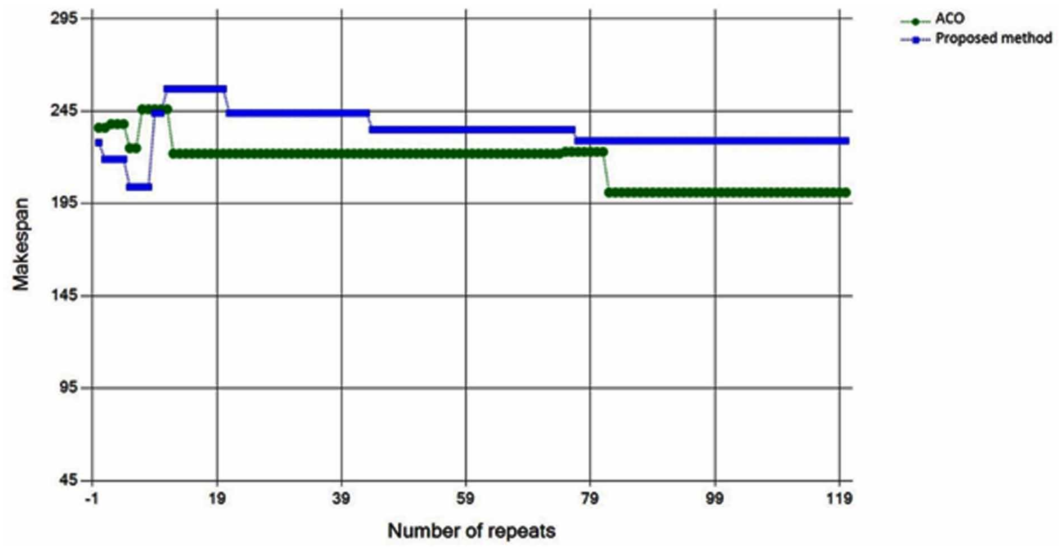
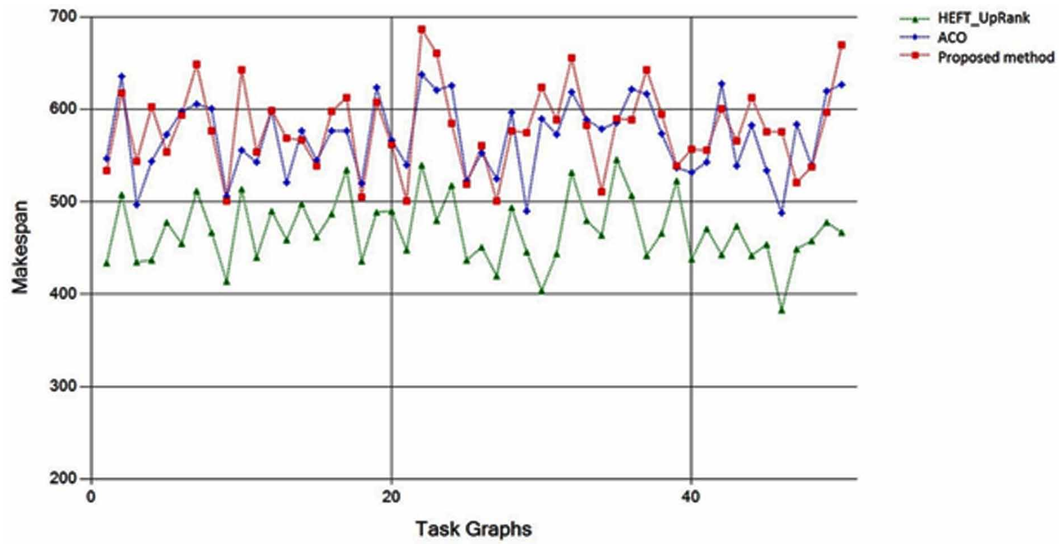


Figure 12. Makespan of 50 random generated DAG (number of tasks = 50, number of resources = 6, number of population = 200)



proposed method in this graph is better than the other two methods in terms of energy consumption and makespan shows better performance than the HEFT-upward rank method.

5. CONCLUSION AND FUTURE WORK

This paper studies the task scheduling problem with priority limitation. In the proposed method, the total execution time and energy consumption are considered as two factors under investigation.

Figure 13. Energy consumption for 50 random generated DAG (number of tasks = 50, number of resources = 6, number of population = 200)

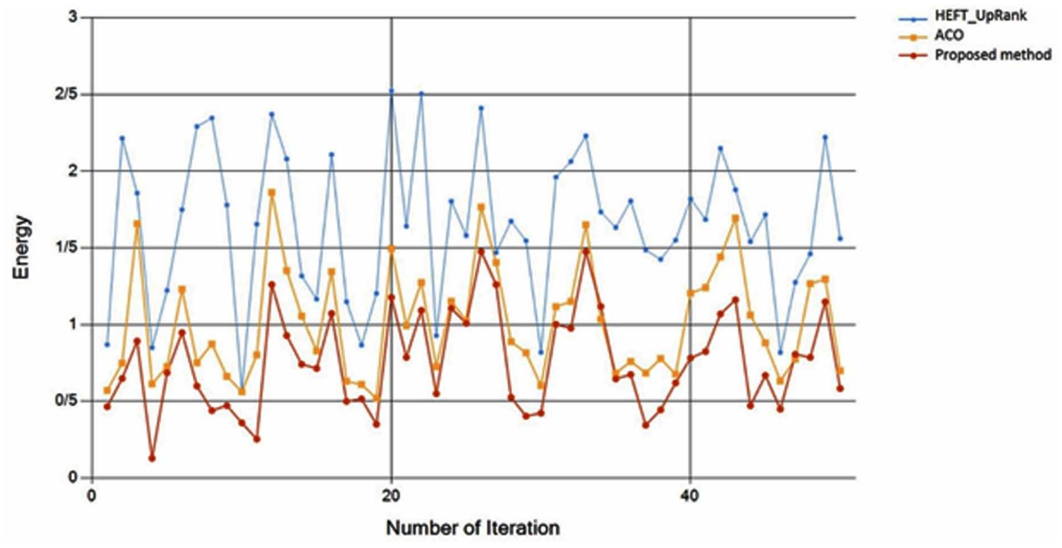
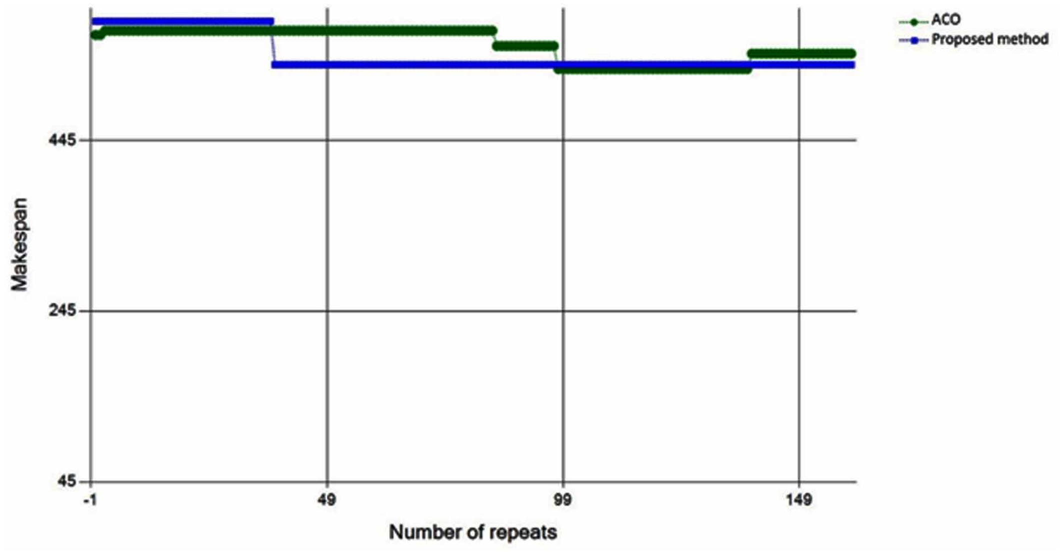


Figure 14. Convergence of the best solutions in each repeat of selected sample of 50 graphs in Figure 12



Considering the importance of these two factors, a new method is presented that combines ACO and cultural algorithm for scheduling the tasks and priorities in green computing. In this way, we tried to take advantages and avoid disadvantages of both algorithms. The results represent that the proposed method outperforms HEFT-upward rank algorithm and ACO in terms of energy consumption and

Figure 15. Makespan of the DAG in Figure 1 using different methods

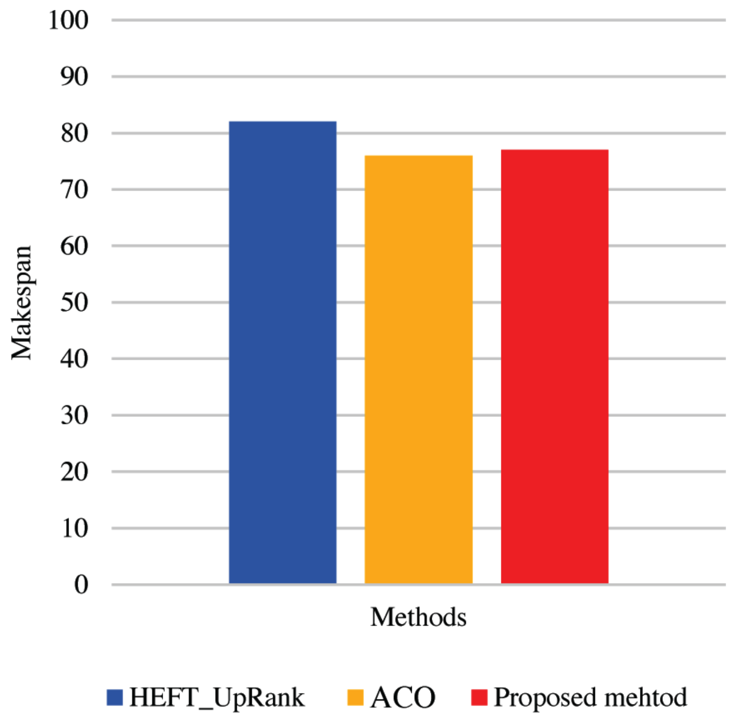
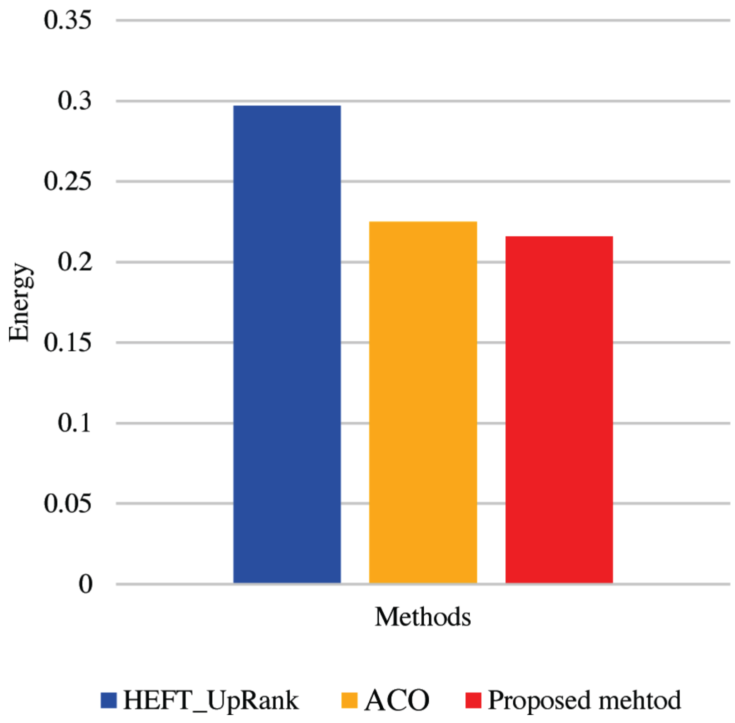


Figure 16. Results of energy consumption of the DAG in Figure 1 using different methods



outperforms the HEFT-upward rank in terms of makespan. However, the proposed method improves the makespan but could not excel from ACO.

The communication overhead, the voltage/ frequency switching overhead and other uncertain parameters in the actual presence of a heterogeneous environment will be discussed in the future research. We plan to use DVFS technique to influence on a server's power efficiency.

ACKNOWLEDGMENT

Nima Jafari Navimipour is a corresponding author on this paper and can be reached at jafari@iaut.ac.ir.

REFERENCES

Abdelhalim, E. A., & El Khayat, G. A. (2016). A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA). *Alexandria Engineering Journal*, *55*(2), 1395–1409. doi:10.1016/j.aej.2016.02.017

Alami Milani, B., & Jafari Navimipour, N. (2016). A comprehensive review of the data replication techniques in the cloud environments. *Journal of Network and Computer Applications*, *64*(C), 229–238. doi:10.1016/j.jnca.2016.02.005

Alkhanak, E. N., Lee, S. P., & Khan, S. U. R. (2015). Cost-aware challenges for workflow scheduling approaches in cloud computing environments: Taxonomy and opportunities. *Future Generation Computer Systems*, *50*, 3–21. doi:10.1016/j.future.2015.01.007

Ashouraie, M., & Jafari Navimipour, N. (2015). Priority-based task scheduling on heterogeneous resources in the Expert Cloud. *Kybernetes*, *44*(10), 1455–1471. doi:10.1108/K-12-2014-0293

Aznoli, F., & Navimipour, N. J. (2017). Cloud services recommendation: Reviewing the recent advances and suggesting the future research directions. *Journal of Network and Computer Applications*, *77*, 73–86. doi:10.1016/j.jnca.2016.10.009

Bouarara, H. A., Hamou, R. M., Rahmani, A., & Amine, A. (2014). Application of Meta-Heuristics Methods on PIR Protocols Over Cloud Storage Services. *International Journal of Cloud Applications and Computing*, *4*(3), 1–19. doi:10.4018/ijcac.2014070101

Chiregi, M., & Navimipour, N. J. (2016). Trusted services identification in the cloud environment using the topological metrics. *Karbala International Journal of Modern Science*, *2*(3), 203–210. doi:10.1016/j.kijoms.2016.06.002

Chong, H.-Y., Wong, J. S., & Wang, X. (2014). An explanatory case study on cloud computing applications in the built environment. *Automation in Construction*, *44*, 152–162. doi:10.1016/j.autcon.2014.04.010

Chowhan, S. S., Shirwaikar, S., & Kumar, A. (2016). Predictive Modeling of Service Level Agreement Parameters for Cloud Services. *International Journal of Next-Generation Computing*, *7*(2), 115–129.

Copeland, M., Soh, J., Puca, A., Manning, M., & Gollob, D. (2015). Microsoft Azure and Cloud Computing. In *Microsoft Azure: Planning, Deploying, and Managing Your Data center in the Cloud* (pp. 3–26). Berkeley, CA: Apress; doi:10.1007/978-1-4842-1043-7_1

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66. doi:10.1109/4235.585892

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons. doi:10.1002/9780470512517

Ezugwu, A. E., Buhari, S. M., & Junaidu, S. B. (2013). Virtual machine allocation in cloud computing environment. *International Journal of Cloud Applications and Computing*, *3*(2), 47–60. doi:10.4018/ijcac.2013040105

Habibi, M., & Navimipour, N. J. (2016). Multi-Objective Task Scheduling in Cloud Computing Using an Imperialist Competitive Algorithm. *International Journal of Advanced Computer Science & Applications*, *1*(7), 289–293.

Kang, Y., Zhang, Y., Lin, Y., & Lu, H. (2014). An Improved Ant Colony System for Task Scheduling Problem in Heterogeneous Distributed System. In *Computer Engineering and Networking* (pp. 83–90). Springer. doi:10.1007/978-3-319-01766-2_10

Keshanchi, B., & Navimipour, N. J. (2016). Priority-Based Task Scheduling in the Cloud Systems Using a Memetic Algorithm. *Journal of Circuits, Systems, and Computers*, *25*(10), 1650119. doi:10.1142/S021812661650119X

Keshanchi, B., Souri, A., & Navimipour, N. J. (2017). An improved genetic algorithm for task scheduling in the cloud environments using the priority queues: Formal verification, simulation, and statistical testing. *Journal of Systems and Software*, *124*, 1–21. doi:10.1016/j.jss.2016.07.006

Khan, M. A. (2012). Scheduling for heterogeneous Systems using constrained critical paths. *Parallel Computing*, *38*(4), 175–193. doi:10.1016/j.parco.2012.01.001

Kim, W.-C., & Jo, O. (2016). *Cost-optimized configuration of computing instances for large sized cloud systems*. ICT Express.

Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *Analytical Press*.

Kumar, P. S., Ashok, M. S., & Subramanian, R. (2012). A publicly verifiable dynamic secret sharing protocol for secure and dependable data storage in cloud computing. *International Journal of Cloud Applications and Computing*, *2*(3), 1–25. doi:10.4018/ijcac.2012070101

Mehta, H. K., & Gupta, E. (2013). Economy based resource allocation in IAAS cloud. *International Journal of Cloud Applications and Computing*, *3*(2), 1–11. doi:10.4018/ijcac.2013040101

Milani, A. S., & Navimipour, N. J. (2016). Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications*, *71*, 86–98. doi:10.1016/j.jnca.2016.06.003

Mohammadi, S. Z. N. J. N. (2017). Invalid cloud providers' identification using the support vector machine. *The International Journal of Next-Generation Computing*.

Navimipour, N. J. (2015a). Task Scheduling in the Cloud Computing Based on the Cuckoo Search Algorithm. *International Journal of Modeling and Optimization*, *5*. doi:10.7763/IJMO.2015.V5.434

Navimipour, N. J. (2015b). Task scheduling in the Cloud Environments based on an Artificial Bee Colony Algorithm. In *Proceedings of 2015 International Conference on Image Processing, Production and Computer Science (ICIPCS '15)*, Istanbul.

Navimipour, N. J., & Milani, B. A. (2016). Replica selection in the cloud environments using an ant colony algorithm. In *Proceedings of the 2016 Third International Conference on Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC)* (pp. 105–110). doi:10.1109/DIPDMWC.2016.7529372

Navimipour, N. J., Rahmani, A. M., Navin, A. H., & Hosseinzadeh, M. (2015). Expert Cloud: A Cloud-based framework to share the knowledge and skills of human resources. *Computers in Human Behavior*, *46*, 57–74. doi:10.1016/j.chb.2015.01.001

Reynolds, R. G., & Zhu, S. (2001). Knowledge-based function optimization using fuzzy cultural algorithms with evolutionary programming. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics*, *31*(1), 1–18. doi:10.1109/3477.907561 PMID:18244764

Shao, J., Wang, Q., & Mei, H. (2012). Model based monitoring and controlling for Platform-as-a-Service (PaaS). *International Journal of Cloud Applications and Computing*, *2*(1), 1–15. doi:10.4018/ijcac.2012010101

Sheikholeslami, F., & Navimipour, N. J. (2017). *Service allocation in the cloud environments using multi-objective particle swarm optimization algorithm based on crowding distance*. Swarm and Evolutionary Computation.

Singh, A., Dutta, K., & Singh, A. (2014). Resource allocation in cloud computing environment using AHP technique. *International Journal of Cloud Applications and Computing*, *4*(1), 33–44. doi:10.4018/ijcac.2014010103

Singh, A., & Malhotra, M. (2015). Agent based resource allocation mechanism focusing cost optimization in cloud computing. *International Journal of Cloud Applications and Computing*, *5*(3), 53–61. doi:10.4018/IJCAC.2015070104

Sood, S. K. (2013). A value based dynamic resource provisioning model in cloud. *International Journal of Cloud Applications and Computing*, *3*(2), 35–46. doi:10.4018/ijcac.2013040104

Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S. U., & Li, K. (2016). An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. *Journal of Grid Computing*, *14*(1), 55–74. doi:10.1007/s10723-015-9334-y

Tumeo, A., Pilato, C., Ferrandi, F., Sciuto, D., & Lanzi, P. L. (2008). Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation SAMOS '08* (pp. 142–149). doi:10.1109/ICSAMOS.2008.4664857

Vakili, A., & Navimipour, N. J. (2017). Comprehensive and systematic review of the service composition mechanisms in the cloud environments. *Journal of Network and Computer Applications*, *81*, 24–36. doi:10.1016/j.jnca.2017.01.005

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. doi:10.1016/j.ins.2014.02.122

Yang, Y., & Gu, X. (2014). Cultural-based genetic tabu algorithm for multiobjective job shop scheduling. *Mathematical Problems in Engineering*.

Zhu, F., & Wang, H. (2016). A modified ACO algorithm for virtual network embedding based on graph decomposition. *Computer Communications*, *80*, 1–15. doi:10.1016/j.comcom.2015.07.014

Zuo, L., Shu, L., Dong, S., Zhu, C., & Hara, T. (2015). A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing. *IEEE Access*, *3*, 2687–2699. doi:10.1109/ACCESS.2015.2508940

Poopak Azad received her BS in computer engineering, software engineering, from University College of Daneshvaran, Tabriz, Iran, in 2014 and the MS in computer engineering, software engineering, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2017. Her research interests include Cloud Computing and Fault-Tolerance Software.

Nima Jafari Navimipour received his BS in computer engineering, software engineering, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2007; the MS in computer engineering, computer architecture, from Tabriz Branch, Islamic Azad University, Tabriz, Iran, in 2009; and the PhD in computer engineering, computer architecture, from Science and Research Branch, Islamic Azad University, Tehran, Iran in 2014. He is an Assistance Professor in the Department of Computer Engineering at Tabriz Branch, Islamic Azad University, Tabriz, Iran. He has published more than 100 papers in various journals and conference proceedings. His research interests include Cloud Computing, Social Networks, Fault-Tolerance Software, Computational Intelligence, Evolutionary Computing, and Network on Chip.