Received: 30 March 2022 | Revised: 19 August 2022 | Accepted: 16 September 2022
DOI: 10.1002/cpe.7409

RESEARCH ARTICLE

WILEY

Optimized multi-objective Q-learning with enhanced beetle swarm optimization based scientific workflows scheduling on cloud computing environment

Nivethithai S[1] | Hariharan B[2]

[1]Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, India

[2]Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, India

Correspondence
Hariharan B, Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, India.
Email: hariharanresearch2021@gmail.com

Summary

Cloud technology has raised significant prominence providing a unique market economic approach for resolving large-scale challenges in heterogeneous distributed systems. Through the use of the network, it delivers secure, quick, and profitable information storage with computational capability. Cloud applications are available on-demand to meet a variety of user QoS standards. Due to the large number of users and tasks, it is important to achieve efficient planning of tasks submitted by users. One of the most important and difficult nondeterministic polynomial-hard challenges in cloud technology is task scheduling. To overcome the problem, in this article, optimized multi-objective Q-learning with EBSO-based task scheduling on the cloud is proposed. Q-learning is one of the machine learning algorithms that can solve these types of problems. The proposed approach is divided into two processes: task prioritization and resource selection. In this article, initially, the tasks are prioritized by using the Q-learning algorithm. After the prioritization, the tasks are assigned to resources by using the enhanced beetle swarm optimization (EBSO) algorithm. To achieve this, the multi-objective function is designed based on makespan, cost, and resource utilization. The effectiveness of the suggested method is evaluated using a variety of criteria such as makespan, resource utilization, and cost.

KEYWORDS

cloud computing, enhanced beetle swarm optimization, HEFT, makespan, normalized cost, Q-learning, resource allocation, resource utilization, task prioritization

1 | INTRODUCTION

Cloud computing has gotten a lot of attention in recent years for providing infrastructures for running large-scale systems that require a lot of resources and services.[1,2] The simplest prevalent cloud service paradigm, Infrastructure as a Service (IaaS), delivers resources so that they can be utilized productively. Users can obtain capabilities using a pay-per-use model and a service level agreement (SLA) that establishes the needed Quality of Services (QoS). In reality, a cloud can provide customers with particular capabilities that they can purchase and distribute as needed.[3] Workflow is the simplest essential and widely used cloud model, describing a wide number of analytical purposes.[4] A workflow is represented as a directed acyclic graph (DAG), with nodes representing complex jobs and edges representing task relationships. Workflow scheduling is the process of selecting the most appropriate VMs for each task of a workflow to meet the required QoS. It is extensively viewed to be NP-complete.[5] To do such, communication providers offer a variety of virtual machines (VMs) having varying pricing and characteristics. Different workflow operations demand various planning algorithms in heterogeneous distributed environments to ensure acceptable effectiveness.[6]

The majority of the research focused on homogeneously dispersed organizations, although others focused on heterogeneous systems having various QoS restrictions. Existing methods are more data- or process-oriented than resource-oriented, and they lack effective task-to-resource mapping tools. In addition, characteristics like task transmission time as well as time complexity were thought to be deterministic in classical scheduling.[7,8] Indeed, the numerous aspects involved in scheduling challenges are frequently vague or imprecise in real-world circumstances.[9] Human-defined elements, especially, play a role in scheduling issues. Till recently, investigators have focused a lot of attention on the challenges of dealing with and modeling unknown data.[10] Considering the large expansion of cloud resources, it is difficult to select the optimal capacity to meet the user's QoS requirements while also achieving business objectives that appear to be at odds.[11,12] Optimal resources should be sought, taking into account a variety of qualitative and quantitative parameters that are incompatible.[13] It's crucial to think about how to optimize process scheduling while still meeting the user's QoS requirements. As a result, selecting cloud resources can be thought of as an multi-criteria decision making (MCDM) problem.[14]

Its goal is to choose the best choice among a set of possibilities in the face of several negative decision constraints.[15] The goal of MCDM in workflow scheduling is to evaluate and assess the alternatives depending on the user's QoS restrictions. To overcome workflow scheduling challenges and get better results, most academics have turned to developing multi-objective meta-heuristic algorithms such as ant colony optimization (ACO), simulated annealing,[16] particle swarm optimization (PSO), and genetic algorithm (GA) in recent years. The Q learning technique is used to schedule workflow in cloud computing in this article.

The goal of scheduling is to achieve optimal scheduling of tasks submitted by users and to improve overall performance. The proposed methodology consists of two stages such as task prioritizing and processor allocation. Initially, the tasks are prioritized using HEFT with a Q-learning algorithm. For processor allocation, the EBSO algorithm-based multi-objective function is developed. A good scheduling system should maintain a minimum makespan, and cost and maximize resource utilization. The main contribution of the proposed approach is listed below:

- For task prioritizing, Q-learning with the HEFT algorithm is proposed.
- The multi-objective function is designed to obtain the optimal processor selection on the cloud.
- For processor selection, an enhanced beetle swarm optimization algorithm (EBSO) is proposed. The EBSO is a hybridization of BSO and oppositional-based learning (OBL) strategy.
- The performance of the proposed approach is analyzed in terms of efficiency metrics such as makespan, cost, and resource utilization.

2 | LITERATURE REVIEW

Many researchers had developed scientific workflow scheduling on the cloud. Among them few of the works are analyzed here; Paknejad et al.[17] had investigated to improve the multi-objective co-evolutionary algorithm, named PICEA-g, which was established as an excellent predictive algorithm, where the logistic and tent maps as two chaotic processes were utilized to generate chaotic values. Experiments evaluate that the algorithm outperformed the PICEA in terms of time, cost, and energy consumption.

Doostali et al. created a critical-path-based optimal workflow scheduling approach in 2021.[18] The findings demonstrate whether identifying the essential path reduces workflow execution duration despite preserving a reasonable level of resource cost and consumption.

Chakravarthi et al.[19] investigated technique of order preference by similarity to ideal solution (TOPSIS), a multi-workflow scheduling method based on the MCDM methodology. The ideal capacity within the financial information is determined by a weighted sum of run time, cost, and information transmission time in compliance only with the job specifications.

On the green cloud environment in 2021, Mohammadzadeh et al.[20] improved search speed by making algorithms greedy and using random numbers according to Chaos theory. The goal was to reduce makespan time, task cost, and reduced resource usage while increasing performance. The algorithm has paved a new path for boosting the capabilities of leader-based search.

Using the DVFS mechanism, Rani and Garg et al.[21] were able to reduce the system's energy usage while retaining task implementation dependability. They introduced a fuzzy dominance-based reliable green workflow scheduling method, which uses the fuzzy dominance mechanism to improve the application's dependability and power utilization simultaneously, saving 15%–20% energy.

The meta-heuristic cost-effective firefly-based algorithm (CEFA) was explained by Chakravarthi et al.,[22] which reduces the expenditure of computation within a deadline limitation. To meet the objectives, CEFA takes into account variables such as CPU efficiency fluctuation, delay in acquiring and terminating VMs, and so on. The energy-aware workflow scheduling approach for real experimental workflows in the virtualized cloud architecture was created by Garg et al.[23] in 2021. The scheduling goals resulted in enhanced overall throughput for scheduling workflows as well as increased energy economy. It aided in the reduction of processing duration, such as transmission period, VM construction time, and so on. The monitoring of system usage level when organizing and distributing new VMs has prevented the host overloading issue.

Ziyath and Subramaniyan[24] developed load balancing on the cloud using an improved Q-learning technique. Here, initially, they evaluate the placement value of the tasks in the queue with the present status of the VM in the cluster and reshuffle the task accordingly. This technique has

given better results compared to the other meta-heuristic-based scheduling techniques. Similarly, Li et al.[25] developed a Q-learning algorithm based on task scheduling on the cloud for health- care applications. A Markov decision model was developed to optimize the match process of patients and which has to effectively reduce the cost. In this article, three types of costs were used namely, waiting for time costs, medical costs, and penalty costs. The proposed Q-learning algorithm was effectively scheduling the tasks on the cloud. Moreover, Tong et al.[26] presented scheduling based on a machine learning algorithm. To achieve this concept, the QL-HEFT algorithm was presented. The QL-HEFT is a hybridization of heterogeneous earliest finish time (HEFT) and Q-Learning. The algorithm uses the upward rank (rank) value of HEFT as an immediate reward in the Q-learning framework. This approach consists of two stages namely, the Q-learning-based task sorting phase and processor allocation. This method gives better results in terms of average response time.

3 | DAG MODEL

The dependent standard stable task scheduling is provided in this study. The connecting edges between the pro-work nodes and the work nodes make up an application framework. Due to the characteristics of the application model, it can be specified as a DAG.[1] The DAG is described using Equation (1).

$$G = (T, E, C, W). \tag{1}$$

While T represents the collection of every task, $T = \{T_i | i = 1, 2, \ldots, n\}$, T_i indicates a DAG task, whereas n indicates the maximum quantity of tasks. E is the entire collection of vertices connecting the tasks $E = \left\{ E_{ij} \middle| = (T_i, T_j) \ | T_i, T_j \in T, i < j \right\}$ and T_i represents the immediate predecessor task node T_j. C represents the set of transmission costs between tasks with edge connections, for example, C_{ij} indicates the transmission cost between task T_i and task T_j. W represents the set of weights on the task, it represents the computation costs of the tasks. Figure 1 depicts a simple DAG diagram consisting of eight tasks.

The task node quantity is indicated at the top of each task node in Figure 1. It equates to a portion of the code or instruction in the program and is a small unit in task planning. The average computing cost of tasks across all processors is indicated by the number at the bottom of each task node. A task's average computing cost is estimated using Equation (2) in DAG.

$$\bar{W}_i = \frac{1}{m} \sum_{k=1}^{m} w_{i,k}, \tag{2}$$

where $w_{i,k}$ is time to complete the task, m is overall quantity of processor.

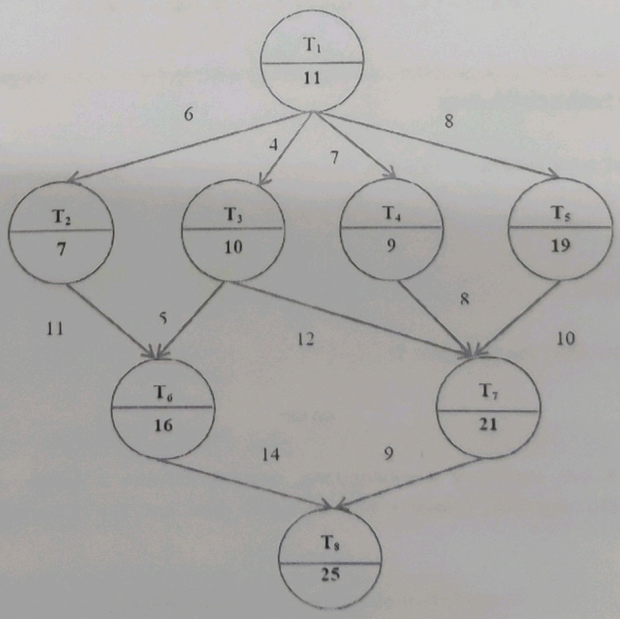

FIGURE 1 Sample directed acyclic graph model

The communication cost between two tasks is calculated using Equation (3).

$$C_{i,j} = L_k + \frac{d_{i,j}}{B_{k,l}}. \tag{3}$$

where L_k represent the communication time of processor P_k, data transmitted between task T_i and task T_j is represented as $d_{i,j}$ and transfer rate among processor P_k and P_l is represented as $B_{k,l}$.

The average communication cost of an edge is calculated using Equation (4).

$$\vec{C}_{i,j} = \vec{L} + \frac{d_{i,j}}{\vec{B}}, \tag{4}$$

where $\vec{L}$ is average communication start-up time of all processors and $\vec{B}$ is average transfer rate.

Usually whenever two jobs are allocated to various processors does there emerge a transmission expense. When several jobs are allocated to identical machines, the expense of transmission among them is zero. In this paper, we set $\bar{C}_{i,j}$ equal to $C_{i,j}$.

We must additionally evaluate a combination of temporal elements while allocating a task to processing to accomplish the work sooner. The initial period a function can be done is while all obvious predecessor tasks have been accomplished as well as every information required by the function has been collected to the processor; this is the period whenever the processor achieves a function while becomes idle. The task's initial started duration, which can be described as the highest of a couple of period intervals, is

$$\mathrm{EST}\left(T_i, P_k\right) = \max\left(\max_{T_q \in \mathrm{pred}(T_i)} \left(FT\left(T_q\right) + c_{q,i}\right), AT\left(P_k\right)\right). \tag{5}$$

While Pred (T_i) is the collection of the task's nearest predecessor tasks T_i, $FT\left(T_q\right)$ is the task's completion duration, T_q, and $AT\left(P_k\right)$ is the processor's latest accessible period. Furthermore, an entry task's EST is 0.

The earliest finish time (EFT) of a task T_i on a processor P_k is identically described.

$$\mathrm{EFT}\left(T_i, P_k\right) = w_{i,k} + \mathrm{EST}\left(T_i, P_k\right). \tag{6}$$

4 | TASK SCHEDULING ALGORITHMS

In this article, for task scheduling combination of HEFT and Q-learning algorithm is used. In this section, we discussed the mathematical expression of both scheduling algorithms.

4.1 | HEFT algorithm for task scheduling

The HEFT algorithm is mainly used for static DAG task scheduling. The algorithm mainly consists of two major steps as task prioritizing phase and the processor selection phase.[8]

4.1.1 | Task prioritizing phase

In this section, the task is prioritized based on the rank value. The

$$RANK^u$$

is calculated using Equation (1). A task containing a greater rank_u quantity provides a greater priority, indicating it gets delivered to the processing quicker. The HEFT algorithm considers task evaluation time and task transmission duration while calculating the rank value.

$$\mathrm{RANK}^u\left(T_i\right) = M_i + \max_{T_j \in \mathrm{succ}(T_i)} \left(\bar{C}_{i,j} + \mathrm{RANK}^u\left(T_j\right)\right), \tag{7}$$

where M_i represent the average computational cost of the task T_i, the set of immediate successors of task T_i is denoted as succ (T_i) and the average communication cost of the edge between tasks T_i and T_j is represented as $\overline{C}_{i,j}$. The exit task is calculated in T_{exit} recycling mode starting from RANK^u and traveling upwards to the task map. In addition, the upward rank value of the exit task $\text{rank}_u(T_{exit})$ is equal to $\overline{w}_{exit}$.

4.1.2 | Processor selection phase

The work is then assigned to a processor based on the existing completing duration method (Equation 6). Each work is allocated to a processor capable of rapidly completing it. Finally, get the MacBook after completing all the tasks.

4.2 | Q-learning algorithm

The Q-learning method is a modeless reinforced learning technique that enables an entity to pick optimum behaviors in a Markov system based on previous information. Q-learning is based on the concept of establishing a response cost purpose for every state-action pair. The quantity is the total response number acquired while operating on the stage. The $Q(s, a)$ gains information automatically through historical knowledge and does not require a thorough understanding of the description of the surroundings a in the state s. As a result, Q-learning has no bearing on the equation. When making selections, the representative only requires evaluating the Q numbers associated with every activity in the state, that is, the operator $Q(s, a)$ could find the best strategy in the state s while taking into account the state follow-up situations. This can make the decision-making approach easier.

Depending on the temporal difference (TD) technique, the Q-learning technique continuously constructs the MDP's response meaning expression. The differential among two consecutive occurrences with identical variables is referred to as TD. Different notification procedures should be used to generate different TD algorithms. In Q-learning, the TD (0) method is developed, and it symbolizes the agents merely changing the value component of the subsequent condition sequentially. The TD (0) updating formula is as follows:

$$V(s) = V(s) + \alpha\left(r + \gamma V\left(s'\right) - V(s)\right), \tag{8}$$

where α is the learning rate $(0 < \alpha \leq 1)$, and γ is the discount factor. The formula uses the difference between the optimal value $r + \gamma V(s')$ and the estimated value $V(s)$ at this moment to correct the estimated value. The revised estimated value $V(s) + \alpha\left(r + \gamma V(s') - V(s)\right)$ is used to replace the original estimated value $V(s)$ and obtain the optimal value at the current moment. The iterative calculation for Q-learning is given in Equation (9).

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha\left(r + \gamma \max_{a} Q_t\left(s', a'\right) - Q_t(s, a)\right). \tag{9}$$

Q-learning utilizes a defined strategy to assure whether the program cumulates following establishing a correct learning rate.[27] The Q-learning algorithm is given in Table 1.

TABLE 1 Q-learning algorithm

Input: Random state process
Output: Q-Table
1. Randomly initialize $Q(s, a)$ for all state-action pairs.
2. **Recap** for each episode k
3. Initialize state s.
4. Recap for each step of the episode
5. Select action a from s using policy derived from Q(e.g., $\varepsilon \in (0, 1)$)
6. Take action a, observe r, s'.
7. Update $Q(s, a)$ with Equation (8)
8. $S \leftarrow s'$
9. Until s is terminal
10. Until convergence ($\lvert Q_k(s, a) - Q_{k-1}(s, a)\rvert < \theta$ (θ is an inf initesimal value))

5 | PROPOSED MODEL

The main objective of the proposed methodology is scientific scheduling on the cloud. For scheduling Q-learning with EBSO algorithm is designed. The proposed approach consists of two stages such as task flow scheduling and resource allocation. For scheduling Q-learning algorithm is developed and for resource allocation EBSO algorithm is developed. The overall structure of the proposed methodology is given in Figure 2.

5.1 | Task flow scheduling using Q-learning

In this stage, initially, the tasks are ordered using the QL-HEFT approach. Initially, the Q-learning algorithm is used to sort the original task order to obtain an optimal task order. To ensure adequate training, we use a random selection strategy across the state s and transfer to the state s'. In addition, to minimize search space and minimize algorithm problems, we ensure that the agent's action choice is legal in each state, that is, follows the dependent relationship between the tasks in the DAG. After the agent selects an action a in state s, we use Equation (8) to update the corresponding Q-value ($Q(s, a)$) in Q-table. In each chapter, all states are traveled, and an updated Q-schedule is obtained. After a limited number of iterations, we obtain a Q-table that is no longer variable in the function, that is, the algorithm is integrated. According to the final integrated Q-schedule, we use the maximum Q-value strategy to obtain the optimal work order (making sure the job selection is legal and selecting the maximum Q-value each time). The algorithm of QL-HEFT + EBSO based scheduling algorithm is given in Table 2.

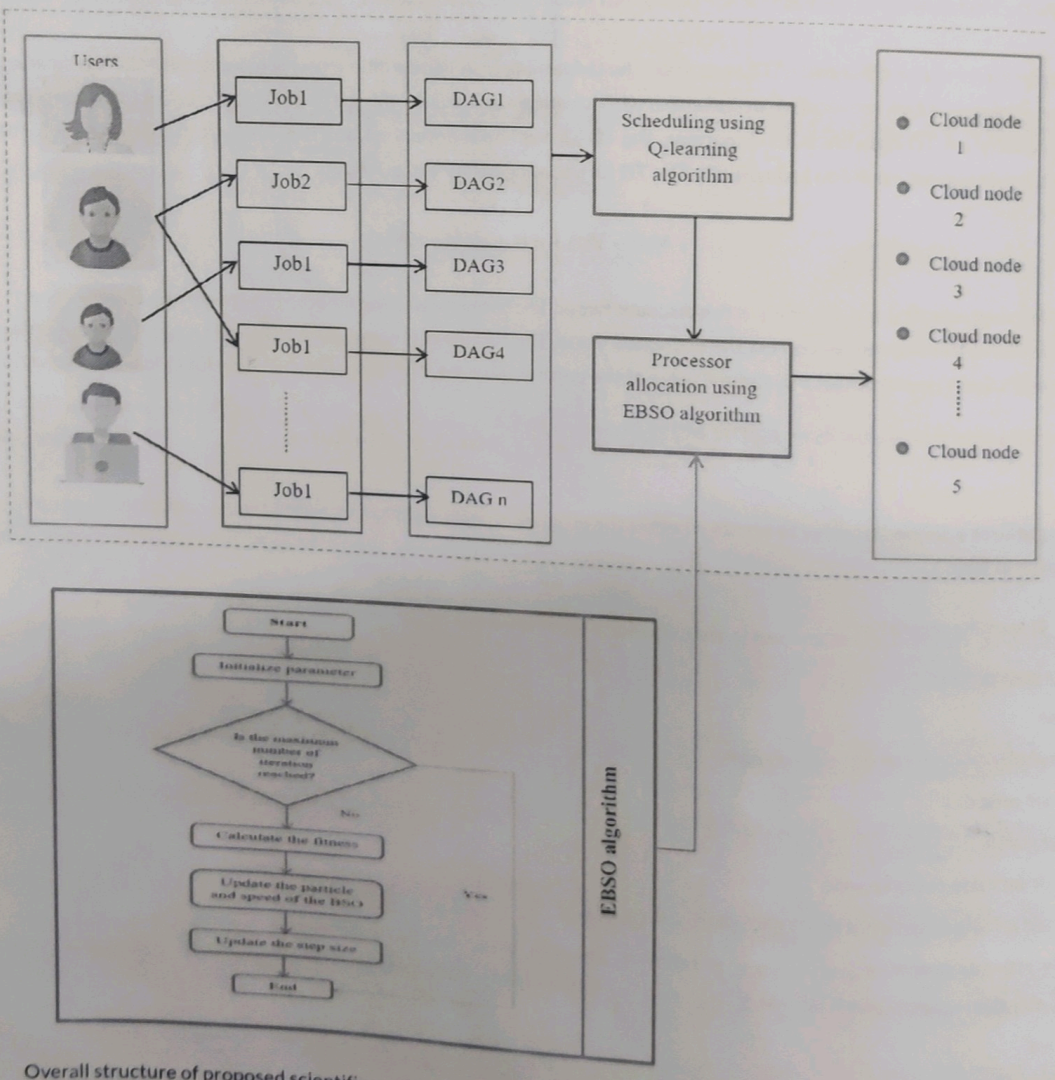

FIGURE 2 Overall structure of proposed scientific workflow scheduling

TABLE 2 Algorithm of QL-HEFT + EBSO-based scheduling

Input: All tasks on DAG//Focus on task size
Output: allocated task
1. Initialize the Q-table to zero
2. Initialize parameters. //including learning rate and discount factor
3. Calculate the immediate reward.//according to RANK^u(Refer Equation 7)
4. Recap for each episode
5. Arbitrarily select an entry task as the current task $T_{current}$
6. Recap for each step of the episode
7. Arbitrarily select a legal task (except for the selected tasks) as the next task T_{next}
8. Update $Q(T_{current}, T_{next})$ by Equation (9)
9. $T_{current} \leftarrow T_{next}$
10. Until T_{next} is terminal
11. Obtain a task order according to the updated Q-table
12. Allocate a processor for each task using EBSO algorithm (refer to Table 3)
13. Obtain the best solution
14. Until convergence (minimum fitness)

TABLE 3 Processor allocation using EBSO algorithm

Input: Maximum number of iterations, number of population size, initialize number VM, number of PM, number of tasks, the parameter of BSO algorithms
Output: The best solution (Minimum fitness)
Start
  Randomly initialize the solution (refer to Figure 3)
    Generate the opposite solution using Equation (10)
      Evaluate the fitness of each solution using Equation (11)
Set cycle 1
Repeat
While ($t < t_{max}$)
  for i = 1:N
  Update the right (S_{RS}^{t+1}) and left (S_{LS}^{t+1}) antenna position using Equation (12)
    Update the incremental function ξ_{js}^{t+1} using Equation (23)
      Update the beetle speed Q_{js}^{t+1} using Equation (22)
        Update the next position of the beetle $S_i(t+1)$ using Equation (21)
          Update the step factor using Equation (25)
end for
$t = t + 1$
end while
stop
output: Best solution

5.2 | Processor selection using an enhanced beetle swarm optimization algorithm

Processor selection is an important process for workflow scheduling. The proper processor selection process can reduce the makespan, cost, and maximize resource utilization. For processor selection, in this article EBSO algorithm is used. The EBSO is a combination of OBL and BSO. Wang and Yang[28] presented the BSO method, which is a revolutionary meta-heuristic optimization approach. It's concept is based on a recreation of the behavior of birds on a beetle. The BSO algorithm provides the benefits of requiring minimal variables, therefore, being straightforward to construct, as well as being inexpensive, flexible, and durable, having minimally susceptible to input measurement limitations, and being able to tackle difficult nonlinear optimization problems. The following is a step-by-step guide to selecting a processor;

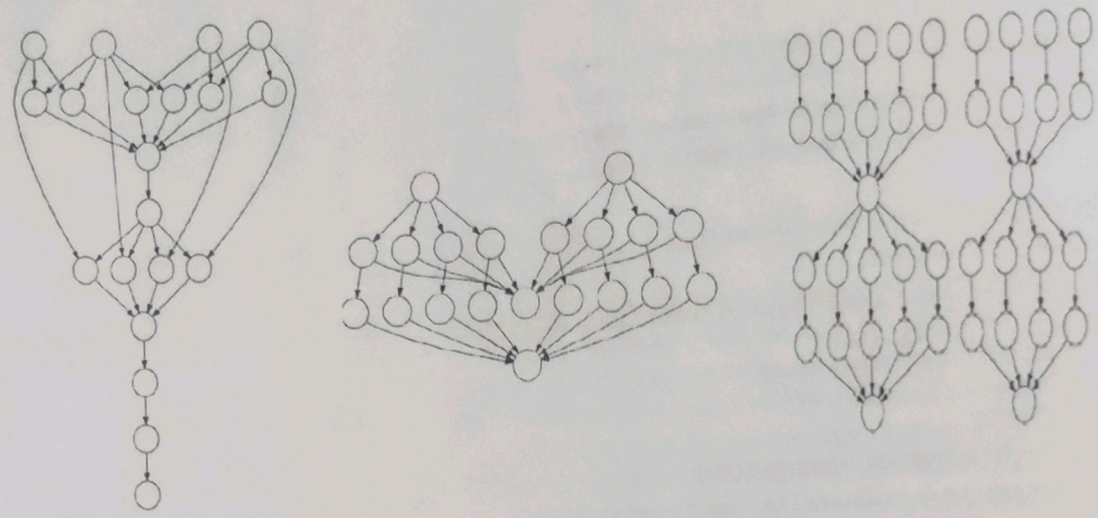

FIGURE 3 Workflow structure (A) Montage, (B) CyberShake, and (C) LIGO

TABLE 4 Solution encoding

PM1			PM2			PM3		
VM_1^1	VM_2^1	VM_3^1	VM_1^2	VM_2^2	VM_3^2	VM_1^3	VM_2^3	VM_3^2
1	0	1	1	1	1	1	1	0

Step 1-Solution encoding: For optimization issues, the encoding of the method is the first step. The work is initially assigned carelessly to the PM. The system includes several activities, as well as PM and VM. To be more exact, the task has three unique characteristics: size, energy, and memory. Every PM also includes the number of virtual machines, RAM, and data transmission capabilities. The jobs are assigned to the VM based on the characteristics, limiting the cost, time, and quantity of transfers. Examine the following scenario: the solution contains three PMs, each of which contains three VM. The encoding of the solution design is shown in Figure 3.

In Table 4, the seven tasks are assigned to VM_1^1, VM_3^1, VM_1^2, VM_1^2, VM_2^2, VM_1^3, and VM_2^3, respectively.

Step 2-Opposite solution generation: Following the response activation, the OBL approach is used to create the opposing solutions. To improve the capacity to search, alternative approaches are applied. Due to increasing searching ability, the processing time will be reduced. Where $S \in [a, b]$ is a real number and the opposite solution $\bar{S}$ is estimated as follows:

$$\bar{S} = a + b - S. \tag{10}$$

Step 3-Fitness calculation: The fitness metric is calculated using the variables makespan, cost, and resource consumption. For every response, the fitness is assessed. To find the best solution, the fitness metric is employed. The minimization function is used for fitness for this article. Equation (11) is used to calculate the fitness function.

$$\text{Fitness} = \min\left(\text{Makespan}, \text{Cost}, {}^{1}/_{\text{Resource utilization}}\right). \tag{11}$$

Makespan: The duration it requires to finish a workflow is used to estimate Makespan. The duration takes to complete the departure task of a resolution makespan can be calculated thanks to the DAG's architecture. Consider the start time ST^{start} and end time ET^{end} of the task TA_a. Task TA_a start time depends on all its parental task completion time and data transfer time between parent and T_i. The start time is calculated using Equation (12).

$$ST^{start}(TA_a) = \begin{cases} 0 & TA_a = T_{entry}, \\ \max\limits_{TA_b \in Parent(TA_a)} \left\{ CT^{com}(TA_b) + TT^{trm}_{a,b} \right\} & \text{otherwise}, \end{cases} \tag{12}$$

where $CT^{com}(TA_a)$ represent the completion time of the task TA_a. The completion time is calculated using Equation (13).

$$CT^{com}(TA_a) = ST^{start}(TA_b) + ET^{exe}_{b,k}. \tag{13}$$

Once we find out the $ST^{start}(TA_a)$ and $CT^{com}(TA_a)$, the makespan of workflow is calculated using Equation (14).

$$\text{Makespan} = CT^{com}(T_{exit}). \tag{14}$$

Cost: Users of cloud services are adjusted to rely on a pay-per-use structure, in that they must pay depending on how long they have used the product. The actual expense of the task t_a is estimated using three costs: processing costs, data transport costs, and storage costs. The cost of the execution task T_i is calculated using Equation (15).

$$\cos t^{PC}_a = T^{exe}_{a,k} \times C^{PC}_k. \tag{15}$$

$T^{exe}_{a,k}$ is the execution time of a task TA_a. C^{PC}_k is processing cost.

The data transfer price among tasks TA_a and their child is evaluated using Equation (16)

$$\text{Cost}^{BW}_a = \sum_{TA_b \in T : TA_a \in \text{Parent}(TA_b)} T^{trm}_{b,a} \times C^{BW}_k, \tag{16}$$

where C^{BW}_k demonstrate the cost of using bandwidth. The storage cost of the task TA_a is evaluated using Equation (17).

$$\text{Cost}^{S}_a = \left(T^{exe}_{a,k} + \max_{TA_b \in \text{Parent}(TA_b)} T^{trn}_{a,b} \right) \times C^{s}_k. \tag{17}$$

C^{s}_k is the instance type P_k storage cost.

The total cost is calculated using Equation (18).

$$\text{Cost} = \sum_{t_i \in T} \text{Cost}^{PC}_i + \text{Cost}^{BW}_i + \text{Cost}^{S}_i. \tag{18}$$

Resource utilization: The allocation of assets VM_k is the same as the number of tasks that the VM has executed up to its finishing period. The VM_k is calculated using Equation (19).

$$U_k = \frac{\sum_{TA_a \in A} MI(TA_a)}{\sum_{TA_a \in A} \left(T^{exe}_{a,k} + \max_{TA_b \in \text{Parent}(TA_a)} T^{trm}_{a,b} \right)}, \tag{19}$$

where A represents the group of tasks and $MI(TA_a)$ denotes the task size TA_a denotes the instructions. The mean usage is calculated using Equation (20).

$$\text{Utilization} = \frac{\sum_{VM_k \in \vartheta} U_k}{|\vartheta|}, \tag{20}$$

where ϑ is the set of instances and $|\vartheta|$ shows the cardinality of it.

Step 4-Updation: After the fitness calculations, the solutions are updated using BSO operations. The position of the beetle is updated their position using Equation (21).

$$S_i(t+1) = S_i(t) + \lambda Q_i(t) + (1-\lambda)\varepsilon_i(t), \tag{21}$$

where t is current number of iterations, Q_i is speed of beetles, ε_i is beetle position movement, and λ is positive constant.

The beetle's speed is estimated using (22)

$$Q_i(t+1) = \omega Q_i(t) + c_1 r_1 (U_i(t) - S_i(t)) + c_2 r_2 (U_{Gi}(t) - S_{Gi}(t)), \tag{22}$$

where c_1, c_2 is two optimistic constants, r_1, r_2 is the arbitrary functions inside the range [0, 1], ω is inertia weight, $U_i(t)$ is local best solution, and $U_{Gi}(t)$ is global best solution.

This work includes the use of decreasing inertia weight, which is defined as follows

$$\omega = \frac{\omega_{max} - \omega_{max} - \omega_{min}}{t_{max}} * t. \tag{23}$$

where $\omega_{min}, \omega_{max}$ is the minimum and maximum value of ω, t, t_{max} is the current and the greatest quantity of iterations.

Where ξ represents the incremental function and is defined in (24).

$$\xi_i(t+1) = \delta(t) * Q_i(t) * sign\left(f\left(S_{rs}(t)\right) - F\left(S_{ls}(t)\right)\right). \tag{24}$$

Using Equation (16), we can increase the updated solution to a high dimension. Here δ represent the step size. The step size is calculated using Equation (25).

$$\delta(t) = eta * \delta(t-1), \tag{25}$$

where *eta* is the viable step factor that is 0.95. The left and the right antenna update their position using Equation (26).

$$\begin{aligned} S_{RS}(t+1) &= S_{RS}(t) + Q_i(t) * D/2. \\ X_{LS}(t+1) &= S_{LS}(t) - Q_i(t) * D/2. \end{aligned} \tag{26}$$

Step 5-Stopping criteria: Whenever the greatest fitness number or the maximal quantity of iterations is reached, the algorithm stops working. The best solution is used for the processor selection process. The EBSO based processor selection algorithm is given in Table 3.

6 | RESULTS AND DISCUSSION

This section discusses the results obtained from the proposed EBSO algorithm-based scientific workflow task scheduling. The proposed approach is performed on a PC having 4GB storage and an Intel Core i5 processor running Windows 10. The proposed approach is implemented in Java. Montage, CyberShake, and LIGO are three forms of procedures utilized for experimental evaluation. Figure 3 illustrates the workflow organization. The VM specification is presented in Table 5 and Amazon EC2 VM instance specification is listed in Table 6. Moreover, parameter range of EBSO algorithm is presented in Table 7.

TABLE 5 VM specifications

Type	Processing capacity	Processing cores	RAM (MB)	Bandwidth (Mbps)	Storage
A	1000	1	512	512	512
B	2000	2	1024	1024	1024
C	3000	4	2048	2048	2048

TABLE 6 Amazon EC2 VM instance specification

Instance type	Core speed	Processing cores	RAM (GB)	Storage (GB)	Cost per hour ($)
m1.small	1	1	1.6	150	0.5
m1.large	5	3	7.6	900	0.22
m1.xlarge	9	5	16	1710	0.51
c1.medium	6	3	1.8	370	0.29
c1.xlarge	21	9	7.2	1710	1.20

TABLE 7 Parameter of EBSO algorithm

Parameters	
Iteration	Range
eta	50
δ	0.95
C_1, C_2	0.2
r_1, r_2	0.2
λ	0.1
	0.15

6.1 | Experimental results

The experimental outcomes gained through the proposed methodology are examined in this section. The proposed approach's main goal is to schedule the work and assign it to the processor. The Q-learning and multi-objective based EBSO algorithm is utilized to achieve the goal. The makespan, cost, and resource usage are all factors in the multi-objective function.

6.1.1 | Experimental results based on makespan

Makespan is an essential scheduling variable. The makespan varies according to the workflow scheduling mechanism. This section examines the outcomes of several workflow models.

In Figure 4, the effectiveness of the recommended approach is analyzed using makespan. When analyzing Figure 4, recommended approach reached the makespan of 40 s for a small instance, 47 s for a medium instance, and 54 s for a large instance type. The consequences illustrate indicate our recommended technique has a shorter makespan than BSO, PSO, and GA-based task scheduling. This is due to the adaptation of EBSO. Besides, the BSO algorithm has better results compared to the other two methods. Besides extracting alternatives inside a specified circumference of efficient focuses as well as replacing individuals with innovative probabilistic methods, the proposed EBSO can avoid entanglement in the local optimal region of the search space. Figure 5 shows the outcomes of CyberShake workflow-based experiments. The makespan is a crucial variable. The makespan varies depending on the task flow. In comparison to existing approaches, the suggested method has the shortest makespan (see Figure 5). Figure 6 shows the LIGO workflow-based task scheduling experimental results. Here, the experimental analysis is carried out based on three instances. Compared to three instances, a large instance takes the maximum makespan. When analyzing Figure 6, the proposed method is taken a maximum makespan of 700 s. From Figures 4–6, the proposed method is taken a maximum makespan of 55, 120, and 700 s for Montage, CyberShake, and LIGO, respectively.

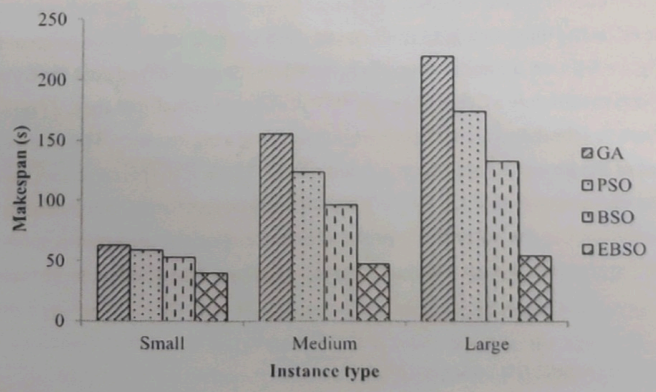

FIGURE 4 Comparative analysis based on makespan for Montage

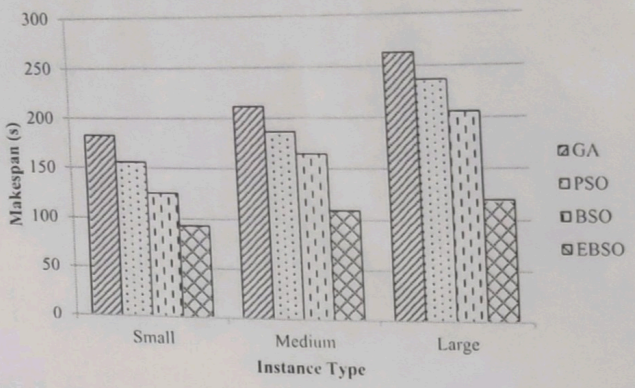

FIGURE 5 Comparative analysis based on makespan for CyberShake

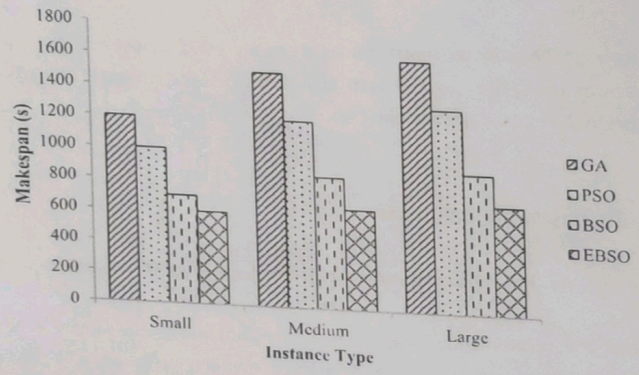

FIGURE 6 Comparative analysis based on makespan for LIGO

6.1.2 | Experimental results based on cost

We evaluate the effectiveness of its suggested technique depending on normalized cost in this section. Each virtual machine has a different cost of processing tasks especially in a heterogeneous cloud environment. The normalized rate of workflow performance is given in Equation (27).

$$NC = \frac{\text{Cost}_w}{\text{Cost}_w^{chp}}, \tag{27}$$

where Cost_w represent the overall cost of the workflow and Cost_w^{chp} represent the cost of all tasks present in the workflow on the cheapest instance, respectively. The experimental results obtained by three workflows are explained below:

In Figure 7, the effectiveness of the recommended approach is analyzed based on the normalized cost for the Montage workflow. When looking at Figure 7, the result shows that the NC is the most similar to BSO-based scheduling. However, our proposed EBSO is slightly superior to BSO. This is related to EBSO's OBL strategy. Figure 8 shows a comparative study based on the normalized cost for CyberShake. When looking at Figure 8, the recommended approach used 14 dollars to schedule a CyberShake workflow utilizing a small instance, 27 dollars to schedule a CyberShake workflow using a medium instance, and 35 dollars to schedule a CyberShake workflow using a big instance. The price varies depending on the situation. Based on normalized cost, the effectiveness of the proposed approach is examined in Figure 9. In this case, too, the proposed approach outperformed the existing approach. The consequences demonstrate that QL + EBSO significantly outperforms that evaluate strategies for simple, middle, and huge workflows, indicating that our methodology has the potential to protect further regions of the optimal solutions. The explanation for this is that our methodology prevents it from becoming trapped at local minimum places and allows it to find borders efficiently.

6.1.3 | Experimental results based on resource utilization

The effectiveness of the proposed technique is evaluated in this section based on resource utilization. The multi-objective function is constructed in this article depending on three variables: makespan, cost, and resource consumption.

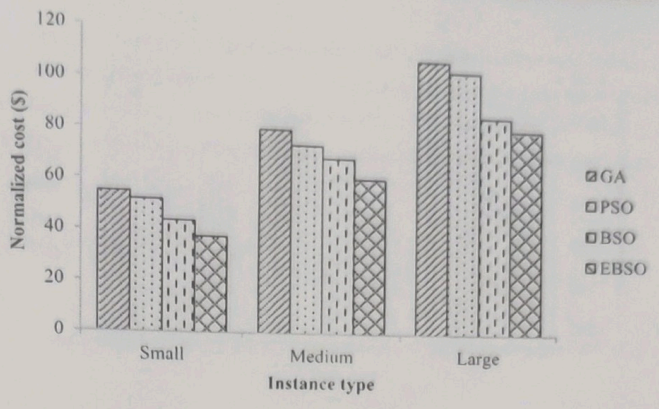

FIGURE 7 Comparative analysis based on the normalized cost for the Montage

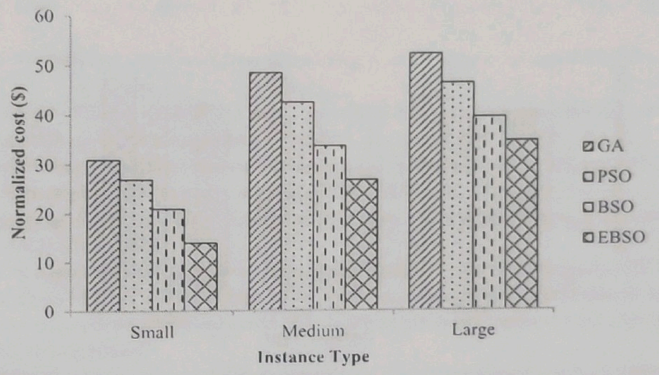

FIGURE 8 Comparative analysis based on the normalized cost for CyberShake

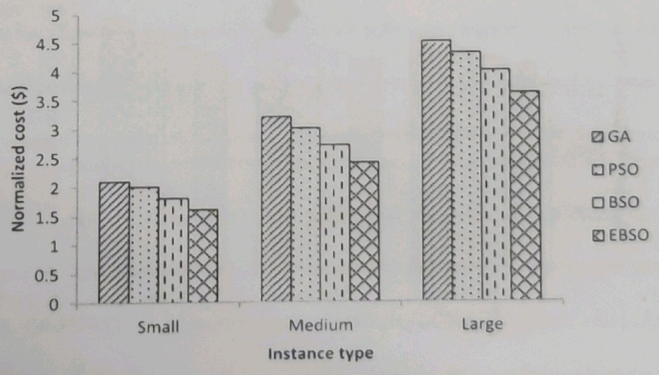

FIGURE 9 Comparative analysis based on the normalized cost for LIGO

Figure 10 shows the performance of recommended approach using resource utilization. A good resource allocation mechanism should make the most of available resources. Figure 10 shows that the proposed strategy used more resources than the other methods for scheduling the Montage workflow model. This suggests that the suggested EBSO is well suited to the scheduling procedure. Similarly, the suggested method achieved superior results in Figures 11 and 12. When compared to the other four techniques GA-based scheduling got the worst outcome. Overall, from the test results, it is clear that our approach works best in reducing makespan and cost as well as increasing the utilization of resources.

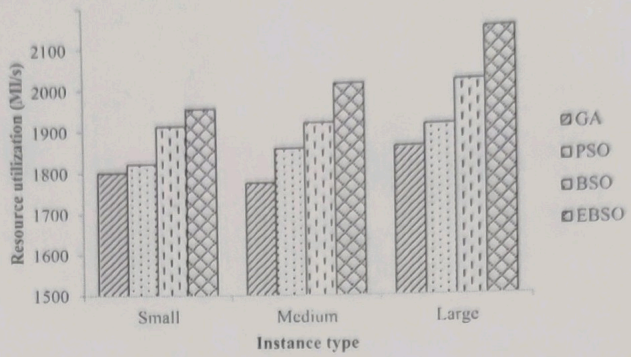

FIGURE 10 Comparative analysis based on resource utilization for Montage

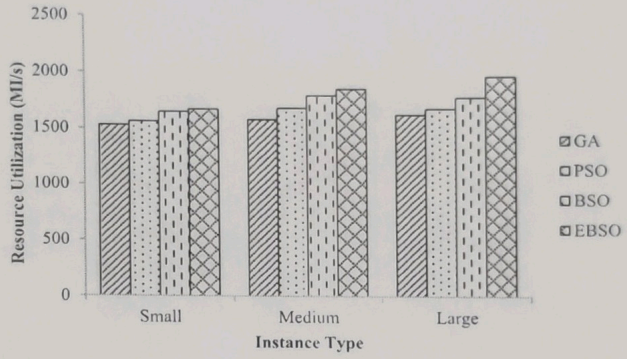

FIGURE 11 Comparative analysis based on resource utilization for CyberShake

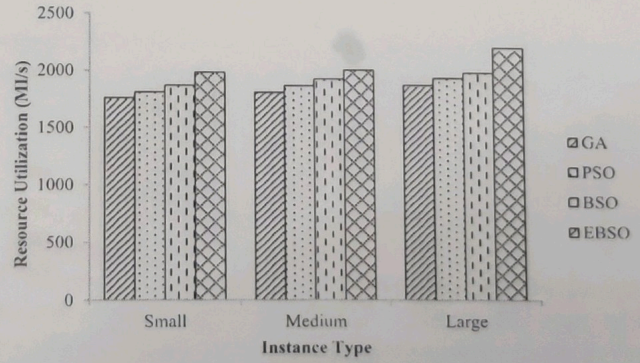

FIGURE 12 Comparative analysis based on resource utilization for LIGO

7 | CONCLUSION

An efficient scientific workflow scheduling has been explained in this article. To achieve this concept, two algorithms have been developed such as QL-HEFT and EBSO. The proposed QL-HEFT has been applied for task prioritizing and EBSO algorithm has been applied for processor allocation. Initially, the task has been prioritized. After the prioritizing, the processor allocated to each task by considering makespan, cost, and resource utilization. The proper task prioritizing and processor selection has decreased the makespan. For experimental analysis, three types of workflows have been analyzed namely, Montage, CyberShake, and LIGO. The performance of the proposed approach has been analyzed based on different metrics. According to the results, for scheduling, our proposed approach takes a minimum makespan of 54 s for Montage, 120 s for CyberShake, and 700 s

for LIGO for the larger instance. The results, clearly show, our proposed approach attained better results compared to the other methods. There are some limitations to our study that present opportunities for future research. For large-scale task optimization, the proposed algorithm has some issues such as an excessively large Q-table producing an excessive update time. In future work, we will consider using a deep neural network to simulate the Q-value methods to solve this problem and combine deep reinforcement learning methods with task scheduling.

CONFLICT OF INTEREST

The corresponding author states that there is no conflict of interest.

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

ORCID

Hariharan B https://orcid.org/0000-0002-8451-8485

REFERENCES

1. Meena J, Kumar M, Vardhan M. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. *IEEE Access*. 2016;4:5065-5082.
2. Chen ZG, Zhan ZH, Lin Y, et al. Multi-objective cloud workflow scheduling: a multiple populations ant colony system approach. *IEEE Trans Cybernet*. 2018;49:1-15.
3. Stavrinides GL, Karatza HD. An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations. *Future Gener Comput Syst*. 2019;96:216-226.
4. Balamurugan S, Saraswathi S. Energy-aware workflow scheduling algorithm for the deployment of scientific work- flows in cloud. In: Satapathy S, Bhateja V, Das S, eds. *Systems and Technologies, Smart Innovation*. Springer; 2018:153-162.
5. Mohanapriya N, Kousalya G, Balakrishnan P, Pethuru Raj C. Energy efficient workflow scheduling with virtual machine consolidation for green cloud computing. *J Intell Fuzzy Syst*. 2018;34(3):1561-1572.
6. Li Z. Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds. *IEEE Trans Serv Comput*. 2019;11(4):713-726.
7. Zhou J, Wang T, Cong P, Lu P, Wei T, Chen M. Cost and makespan-aware workfow scheduling in hybrid clouds. *J Syst Architect*. 2019;100:101631.
8. Saeedi S, Khorsand R, Bidgoli SG, Ramezanpour M. Improved many-objective particle swarm optimization algorithm for scientifc workfow scheduling in cloud computing. *Comput Ind Eng*. 2020;147:106649.
9. Wu F, Wu Q, Tan Y, Li R, Wang W. PCP-B2: partial critical path budget balanced scheduling algorithms for scientific work-flow applications. *Future Gener Comp Syst*. 2016;60:22-34.
10. Haidri RA, Katti CP, Saxena PC. Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing. *J King Saud Univ*. 2020;32:666-683.
11. Iranmanesh A, Naji HR. DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. *Clust Comput*. 2021;24:667-681.
12. Wang Y, Guo Y, Guo Z, Baker T, Liu W. Closure: a cloud scientific workflow scheduling algorithm based on attack–defense game model. *Future Gener Comput Syst*. 2020;111:460-474.
13. Xu X, Dou W, Zhang X, Chen J. EnReal: an energy-aware resource allocation method for scientific workflow executions in cloud environment. *IEEE Trans Cloud Comput*. 2016;4(2):166-179.
14. Garg N, Singh D, Goraya MS. Energy aware hardware and software approaches in cloud environment. *Int J Comput Sci Commun Netw*. 2017;7(3):66-69.
15. Saeedi S, Khorsand R, Bidgoli SG, Ramezanpour M. Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing. *Comput Ind Eng*. 2020;147:106649.
16. Pham T-P, Fahringer T. Evolutionary multi-objective work-flow scheduling for volatile resources in the cloud. *IEEE Trans Cloud Comput*. 2022;10:1780-1791.
17. Paknejad P, Khorsand R, Ramezanpour M. Chaotic improved PICEA-g-based multi-objective optimization for workflow scheduling in cloud environment. *Future Gener Comput Syst*. 2021;117:12-28.
18. Doostali S, Babamir SM, Eini M. CP-PGWO: multi-objective workflow scheduling for cloud computing using critical path. *Clust Comput*. 2021;24(4):3607-3627.
19. Chakravarthi KK, Shyamala L. TOPSIS inspired budget and deadline aware multi-workflow scheduling for cloud computing. *J Syst Archit*. 2020;114:101916.
20. Mohammadzadeh A, Masdari M, Gharehchopogh FS. Energy and cost-aware workflow scheduling in cloud computing data centers using a multi-objective optimization algorithm. *J Netw Syst Manag*. 2021;29(3):1-34.
21. Rani R, Garg R. Reliability aware green workflow scheduling using ε-fuzzy dominance in cloud. *Complex Intell Syst*. 2022;8:1425-1443.
22. Chakravarthi KK, Shyamala L, Vaidehi V. Cost-effective workflow scheduling approach on cloud under deadline constraint using firefly algorithm. *Appl Intell*. 2021;51(3):1629-1644.
23. Garg N, Singh D, Goraya MS. Energy and resource efficient workflow scheduling in a virtualized cloud environment. *Clust Comput*. 2021;24(2):767-797.
24. Ziyath S, Subramaniyan S. An improved Q-learning-based scheduling strategy with load balancing for infrastructure-based cloud services. *Arab J Sci Eng*. 2021;47:1-9.
25. Li Y, Wang H, Wang N, Zhang T. Optimal scheduling in cloud healthcare system using Q-learning algorithm. *Complex Intell Syst*. 2022;1-16.
26. Tong Z, Deng X, Chen H, Mei J, Liu H. QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment. *Neural Comput Appl*. 2020;32(10):5553-5570.

27. Nie J, Haykin S. A Q-learning-based dynamic channel assignment technique for mobile communication systems. *IEEE Trans Veh Technol*. 1999;48(5):1676-1687.
28. Wang T, Yang L. Beetle swarm optimization algorithm: theory and application. arXiv Preprint. arXiv:1808.00206; 2018.

How to cite this article: S N, B H. Optimized multi-objective Q-learning with enhanced beetle swarm optimization based scientific workflows scheduling on cloud computing environment. *Concurrency Computat Pract Exper*. 2022;e7409. doi: 10.1002/cpe.7409