Journal of Optimization in Industrial Engineering

Vol.18, Issue 1, Winter & Spring 2025, 233-243

Research Article

LBO-HFA: A method of simulation to load balancing optimization using the hybrid firefly algorithm

Sana Booshehrian [1], **Ehsan Amiri** [1,*], **Javad Mohammadi Madavani** [2]

1. Department of Computer Engineering, Jahrom University, Jahrom, Iran
2. Department of Computer Engineering, Islamic Azad University, Larestan Branch, Larestan, Iran

https://doi.org/10.71720/joie.2025.1126308

Received: 14 July 2024
Revised: 15 March 2025
Accepted: 28 April 2025

Abstract

The main goal of cloud computing is to achieve higher throughput on a large scale. Load balancing is always a challenge and requires a distributive solution. The response time criterion and energy consumption are evaluated by dynamically transferring the local workload from one machine to another or a less commonly used machine. The main purpose of the load balancing algorithm is to improve the response time by distributing the system's total load. Different algorithms are used in load balancing that can have different parameters. The most important features used are desirability and efficiency. In this report, we optimize the execution time in a set of tasks by examining the load balance parameters and using the Firefly algorithm. The proposed algorithm includes the improved firefly model, which is defined as two parts. The innovation of the present study includes improving the performance of the firefly algorithm and reducing the number of searches in this method, and it has been compared with other optimization algorithms from various aspects. The proposed algorithm enhances the firefly model by improving its performance and reducing the number of searches, as compared to other optimization algorithms. The research results show that the proposed method has a better balance in response time and memory than the GA, NSGA-II, and PSO methods. They also show that the load balance in processor efficiency has a growth of 6% compared to the GA, NSGA-II, and PSO.

Keywords:
Cloud Computing;
Load Balancing;
Firefly Algorithm;
Optimization, Node

Citation:
Booshehrian, S., Amiri, E., & Mohammadi Madavani, J. (2025). LBO-HFA: A method of simulation to load balancing optimization using the hybrid firefly algorithm. *Journal of Optimization in Industrial Engineering*, 18(1), 233-243. https://doi.org/10.71720/JOIE.2025.1126308

* **Corresponding Author:**
Ehsan Amiri
Department of Computer Engineering, Jahrom University, Jahrom, Iran
E-Mail: **e.e.amiri@gmail.com**

1. Introduction

One of the most ironic points about IT is that while computers are much more powerful, they seem slower, and computers are not as fast as they used to be. The reason for this is clear; every day, operating systems and software packages become more sophisticated, their capabilities far exceed the average needs, and more importantly, they require more resources than the resources of a typical computer (Sheikh et al., 2021). Therefore, the rate at which software slows down is greater than when hardware becomes more powerful. Fortunately, there is a solution to this problem. The combination of high-speed Internet and on-demand companies that provide cloud services has created a new world called cloud computing. The basic premise of cloud computing is to have access to services such as software on the Internet instead of on a computer. Programs generally run in a web browser to run the program from any computer with Internet access. A cloud is an abstract image of large, massive networks of unknown size, and we do not know how many processing resources are available (Cardellini et al., 2017). When the demand for computer resources increases, their proper distribution becomes important. If one processing unit has many tasks and the other unit is almost idle, resources are not used well. Also, the time Completion of total tasks can be greatly increased. In general, from a computational point of view, the process of balanced load distribution on processing units is called load balancing (Bhoyar et al., 2015).

Cloud computing is one of the latest developments in information technology, and it is becoming pervasive over time. The cloud computing platform is a fully automated server platform that allows users to purchase telecommunications, dynamic scalability, and system management.

Cloud computing is a model that provides easy access through the network at the user's request to a set of customizable and configurable computing resources such as networks, servers, storage space, applications, and services, that should be managed with minimal need for management. Resources or direct involvement of the service provider should be provided or released immediately. Cloud computing has a three-tier architecture where in the infrastructure layer, we face resource management. In the substrate layer, the possibilities of developing applications on cloud resources are provided. In the software layer, application software is used for the end-user service. If we want to improve the way we manage resources and schedule things according to the issue, we must enter the infrastructure layer. If we want to develop a particular issue in general, we enter the platform layer, and if we want to focus on providing the end service to the user, we must enter the software layer.

The main goal of Cloud computing is to improve distributed resources, combine them to achieve higher throughput, and solve large-scale computational problems (Mahendiran et al., 2012). Load balance and reliability are currently a challenge in cloud computing systems. A distributed solution is always needed, as it is not always possible or cost-effective to maintain one or more idle and inactive servers just to meet certain requirements. Clearly, due to these systems' scale and complexity, the centralized assignment of tasks to specific servers is impossible. To create proper resource management and provide services, we need the load balancer offered to the service provider.

Load balance is a general term used to distribute larger processing loads to smaller processing nodes to improve overall system performance. In a distributed environment, load balancing is a load distribution process between different distributed systems used to improve resource utilization and response time. The idea of a load balancing algorithm should prevent overload and low load on any particular node. In the case of a cloud computing environment, choosing the appropriate algorithm is not easy because it also includes additional barriers such as security, reliability, throughput, and so on. Therefore, the main purpose of the load balancing algorithm in the cloud computing environment is to improve the response time by distributing the system's total load. The algorithm must ensure that no particular algorithm overloads (khaledian et al., 2021). In this case, we used many algorithms to do this research. One of the collective node intelligence algorithms is an algorithm called the firefly algorithm that optimizes the problem.

Section 2 describes the Related Work. In Section 3, the proposed method is presented, and in Section 4, the results and experiments are reviewed. Finally, in Section 5, a general conclusion of the article is given.

2. Related Work

Bhoyar et al. (2015), proposed an algorithm for work scheduling and dynamic configuration of node loads within the grid system in 2015. Within a distributed computing system, processing requests are received randomly from users. A good way to program these requests is to assign them to existing processors. Therefore, all requests may be answered promptly. In Mahendiran et al. (2012), a clustering algorithm that provides a concept of clustering in cloud computing is described. There are many load balancing algorithms in cloud computing (khaledian et al., 2021), and each algorithm has its own advantages and disadvantages. Moreover, depending on the need, one of the algorithms is used. The efficiency of the algorithm can be increased by creating a clustering (Rajeshkannan et al., 2016) of nodes. Each cluster can be considered as a group. The process of creating clusters revolves around the concept of sorting nodes. In this process, the node first selects a neighboring node, a different node, and a sorter. The sorting node establishes a connection with another node of the starting node type, and finally, the sorting node is detached. This process is repeated, and the system efficiency increases due to the high availability of resources. This increase in throughput is due to the efficient use of resources.

An algorithm considers the software dependence on a non-circular directional graph (DG) to set up virtual machines for tasks. The algorithm decides whether the same virtual machine is intended for more than one task. If the execution time is too high at the level of a task, the focus is on reducing it. Reducing runtime can be done by defining more than one

virtual machine for tasks that request the same type of virtual machine, which may increase the cost. On the other hand, if the cost goes up, it reduces it by scheduling more than one task on the same virtual machine, which increases the total execution time. In Dam et al. (2014) an ant-based algorithm is introduced also in Lal et al. (2018) discusses one of the major challenges of cloud computing: load balancing. This means that the workload is distributed dynamically and evenly across multiple nodes.

A good load balancer in cloud computing must adapt its strategy to the changing environment. Avoiding server overhead and balancing the appropriate load between servers can maintain service quality and reduce task completion time. In Raghava et al. (2014), the purpose of this article is to present a new method of load distribution in the cloud computing environment using the bee algorithm. This study aims to properly distribute the load based on minimizing the time of completing tasks on virtual machines. This research is modeled based on 30 and 40 tasks on 5 and 6 virtual machines, and the simulation results are compared with the bee algorithm and the particle swarm optimization algorithm (Kazemi et al., 2024). The results show that the bee algorithm performs better than the particle swarm optimization algorithm and balances the load with the completion time of tasks 146 and 123 in the 30-task mode and 197 and 190 in the 40-task mode on 5 and 6, respectively.

In Li et al. (2018), the Florence method is proposed which uses the glowing insect algorithm. This method consists of three steps: to create the initial population, calculate the timing index, and select the node with the least load. The initial population is considered cloud nodes, and at this stage, the nodes with the lowest load are selected. A list of initial scheduling is then created where the element at the top of the request queue is assigned to the top of the node list. Three types of node processing time characteristics, CPU rate, and memory rate are considered equivalent to the sum of nodes in the scheduling queue. Therefore, the node with the least attractive is selected as a basis for comparison with other nodes. By calculating the Cartesian distance of the other nodes from the node-axis, their attractiveness is calculated. The nodes are then arranged relative to the axis element. The highest queue in the scheduling list is considered the most explicit scheduling queue (Lotfi et al., 2024). This method is improved by the bee algorithm and is used for load balancing. In this method, bee feeding behavior balances the load between virtual machines, which transfers tasks to low-load nodes low-load nodes. In this method, tasks are selected based on priority for migration. Low-priority tasks are candidates for immigration. In this method, each virtual machine's load and the data center's load are calculated. The processing capacity of each virtual machine is measured according to its processing power and bandwidth. Then, by calculating the time required to process the tasks, the standard deviation factor is used with each virtual machine to balance the load. The simulations performed by the authors of this article, show an improvement in the overall execution time and number of tasks.

The method introduced in Maruthanayagam et al. (2014) is a PSO-based method (Khazaei et al., 2021) that considers both types of processing and data-driven tasks. This method also considers bandwidth to reduce data traffic and thus reduce the amount of transmission time. In Keshk et al. (2014), it does not consider idle physical machines as hosts in GA but rather reduces energy consumption. This method includes an optimization (Amiri et al., 2021) model that transfers additional tasks to new homogeneous virtual machines to reduce execution and transfer time. The simulation performed in the paper shows the reduction in the time required for the load balancing process. In Shokry et al. (2022) and Kaur et al. (2017) use the ant colony algorithm for load balancing. Other method is called BLBACO (Dos Santos et al., 2015), which uses the LBACO features to estimate each virtual machine's load and processing power. In the proposed method, the concept of two types of ant movement is used: forward movement and backward movement. Moving forward as a virtual machine search has an overload starting from the virtual machine. Moving backward is also the reverse of that movement. When the search is complete, some of the tasks are assigned from the overloaded virtual machine to the overloaded virtual machine. The ant colony algorithm (Lal et al., 2018) and cuckoo algorithm (Navimipour and Milani, 2016) are used for load balancing. These algorithms consist of three factors: load factor, channel factor, and task migration factor. The load and channel factors are static, and the task migration factor is ant type which is considered a moving factor. The load factor is used as a controller to calculate the load after assigning a new task. The channel agent is used for selection, transfer, and positioning policies. The channel agent activates the task migration agent. These mobile agents go to other data centers and communicate with their load agent to obtain the virtual machines' status and then send them to the channel agent. This algorithm compares each virtual machine's condition with its threshold value.

The proposed method for load balancing called Fuzzy (Xu and Sun, 2016) uses the factor. The agent completes each cycle in two steps. The first step starts moving from the first server and gathers information from all the servers to make the right decision to balance the load (Battula and Vuddanti, 2021). In the second step, the balance balances the servers based on the average cloud load. If the server is overloaded, it sends the tasks to the low-load server, which can be used convolutional neural network (Kazemi et al., 2021). The simulations performed by the authors of the article show a reduction in algorithm execution time, waiting time, and improved throughput (Kanbar and Faraj, 2022). In Negi et al. (2021), the proposed method consists of two parts. The first part is an IWRR scheduler algorithm, which schedules tasks according to the length of tasks and each virtual machine’s current load. Load balancing uses another algorithm called IWRR Balancer (Gerez et al., 2019), which removes tasks from the virtual machine that has the most time required to perform the tasks if a task is completed. The virtual machine is idle and also if the number of current tasks is more than one, it sends them to an idle virtual machine. The simulations performed by the authors show an improvement in the execution time and a reduction in the number of task migrations.

A krill herd algorithm (Hasan et al., 2017) and the firefly algorithm (Shadloo et al., 2017) are used for equitable load distribution. The simplicity of the genetic algorithm (Chung et al., 2004) has led to its use in many optimization problems, but it is an algorithm that searches the problem space globally. It is combined with the gravitational attraction algorithm (Gabhane et al., 2023), which inherently searches the problem space locally to improve load balance and increase the algorithms' efficiency. This study aims to improve a load balancing algorithm based on a genetic algorithm (Gabhane et al., 2023; Kalra et al., 2015). External interactions between tasks are considered and tried to reduce makespan by localizing external connections as much as possible. This method changes the intersection and jumps percentages when the convergence rate slows down to the appropriate solution. The results of experiments clearly show a significant increase in convergence rate to the desired response. In (Miao et al., 2014), a solution based on the firefly algorithm is provided to solve scheduling-dependent tasks. A height-based approach with a random method has been used to create initial solutions in this method. To maintain the interdependence between tasks, transition, and mutation functions are defined. Data transfer between dependent tasks is also considered.

3. Introduced Method

In this research, an algorithm for managing resources and allocating them to requests is presented. In other words, an algorithm for allocating virtual machines for existing tasks that are an optimal model of the firefly algorithm (Miao et al., 2014) is proposed. The ultimate goal of this proposed solution is to obtain a suitable pattern for mapping work to a virtual machine that can minimize response time and completion time without increasing power consumption. Scheduling and resource management in distributed environments, such as cloud computing servers, depends on many parameters. There are different, dynamic, and sometimes interdependent resources and requests in these systems and the system's workload compared to other environments. Given that the most important parameters in cloud computing servers are the parameters based on the time of completion of requests and the appropriate distribution of load, we work on a method that can meet the parameters of response time and completion time of the last task. It represents the proportional distribution of load compared to the previously presented algorithms. To do this, we use a firefly algorithm. This method's proposed approach for creating a load-balanced strategy in the cloud network for programming on nodes is programming by the firefly algorithm (Kashikolaei et al., 2020; Tapale et al., 2020).

The scheduling process is preferred a set of nodes with the least amount of loading. In other words, nodes with a minimum load are preferred to absorb the cloud network's extra process. We consider a virtual machine with three servers in the proposed approach, each server having three nodes. Each node has the characteristics defined by the planning process.

The proposed approach and its general steps can be introduced with the help of a flowchart. According to the proposed model, first, the firefly algorithm's structure will be arranged based on the problem assumptions, and then an initial population will be generated. The best firefly that can select the appropriate load in the network will be considered the optimal answer.

In general, the proposed approach includes three main steps for generating planning on the proposed virtual machine by synchronizing their nodes. The different steps can be as follows:

- Production of the initial population
- Calculation of scheduling index
- Optimal node selection

3.1. Production of the initial population:

The term population refers to a group of cloud service nodes that are requested by users from the server. As it turns out, the server takes a node that is free and separate from the user. Figure 2 shows the cloud system request and response process to generate the initial scheduling list. The server searches for nodes to find free nodes. When the node is obtained, it puts it in the program list, and when the element at the top of the queue is processed, the free node is allocated in response.

The list of basic applications made by a virtual machine with full-cycle processing depends on each node's processing time and the availability of that node. The list of nodes or programs can be displayed in the form of a table, each row containing nodes and columns containing the servers' names.

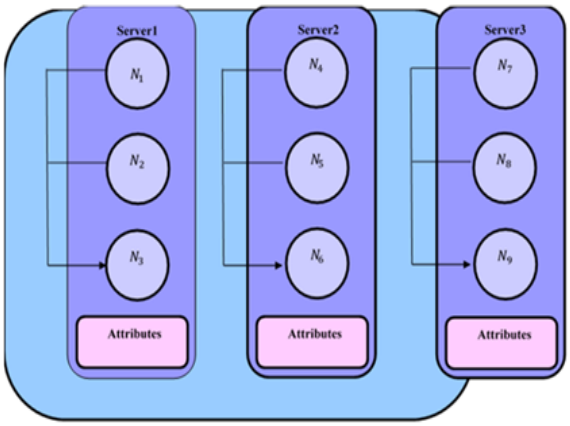


Fig. 1. A basic image of the virtual machine used by the proposed method.

Table 1 contains a list of nodes that are considered as an initial population for the firefly algorithm. Firefly algorithms are applied to a specific population to create an effectively planned strategy in a cloud system by prioritizing load and load balancing.

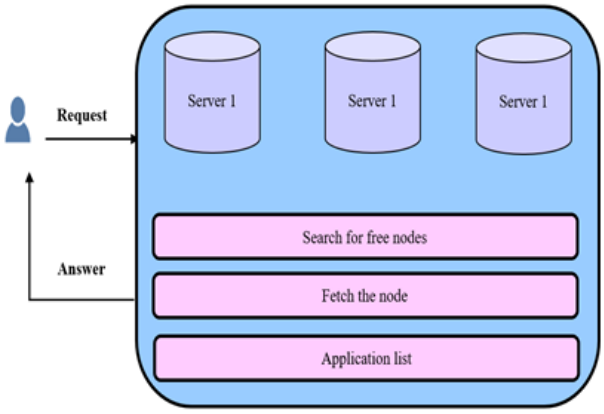


Fig. 2. The cloud system request and response process

Table 1
Preliminary list of the initial population of fireflies

Time (ms)	Server 1			Server 2			Server 3		
1	*N1*	*N2*	*N3*	*N4*	*N5*	*N6*	*N7*	*N8*	*N9*
2	*N1*	*N3*	*N2*	*N9*	*N6*	*N7*	*N5*	*N4*	*N8*
3									
…	*N3*	*N9*	*N1*	*N4*	*N7*	*N6*	*N5*	*N8*	*N2*
N	…	…	…	…	…	…	…	…	…

3.2. Calculation of scheduling index

The scheduling index is one of the main factors influencing the planning process. Therefore, before going into the timing index calculation, we will discuss the initial population's decision parameters. According to the definition of the firefly algorithm, we must have gravity between nodes.

Gravity is based on the dependence of the node on the requests. Decision parameters and the proposed method control gravity. The proposed approaches determine each node's decision parameters, and the parameters are considered as attributes for the nodes. Table 2 shows the nodes and attributes defined by the proposed approach to continue the planning process.

Table 2
Nodes and attributes defined

	Attribute		
Node	**CPU rate**	**Memory rate**	**Processing rate**
N1	$C1$	$M1$	$P1$
N2	$C2$	$M2$	$P2$
…	…	…	…
N_n	C_n	M_n	P_n

CPU rate (C), amount of memory (M), and processing time (P) are considered. The scheduling parameter should be designed to be selected with a minimum load or weight. According to the definitions of the firefly algorithm, the gravity equation is defined as Eq. 1.

$$attr(n_i) = \frac{P_i}{CPU_i + \frac{i}{mem_i}} \quad (1)$$

Here, attr(ni) represents the attraction between the node and the request. Pi indicates the processing time of a particular node, CPUi indicates the node's processor speed, and memi indicates the amount of memory of the nodes. The scheduling index of the above formula is calculated as Eq. 2.

$$SI = \sum_{i=1}^{n} \frac{P_i}{CPU_i + \frac{i}{mem_i}} \quad (2)$$

SI is a scheduling index, and the total number of nodes is in a specific timeline. According to the equation, all scheduling queues in the scheduling list are calculated, and a scheduling list is formed.

C) Selecting the most optimal node

The lowest load node selection processes are inspired by the firefly algorithm and in such a way that at least a separate firefly will have similar characteristics. This goal is inspired by the theory of calculating the distance between nodes in scheduling queues. Before performing the calculations, the node with the minimum values of attr(ni) is found. A node with a minimum attr(ni) is considered an axial point in the queue to calculate distinct nodes. The following Figure 3 shows this method in general.

The flowchart of the proposed method shows which sections have been modified. According to the flowchart, the proposed algorithm consists of the following steps:

Step 1: Get the input information.

At this stage, the information and data on which the problem is applied are prepared and given to the system. This information includes bandwidth, queue length, number of virtual machines, and more. The input data is a number given to the system as an Excel file, and where configuration is required, the required configuration and architecture are provided.

Step 2: Determine the structure of the firefly.

The initial structure of the firefly with one manipulation will strongly affect the execution of its algorithm. The selection of the required parameters for execution was made with multiple repetitions and trial and error.

Step 3: Determine the fitness function.

In this step, a function should be considered to check the correctness of performance and compare between modes. Finally, optimize the function. This function shows the cost of running the firefly algorithm per round. In each round, the parameters obtained from the firefly are given to this function, and the best possible answers are obtained. Moreover, finally, the parameters obtained in the next rounds are used.

Stage 4: Produce the initial population

This step is random so that the initial population and how they are placed in the problem space are random.

Step 5: Choose a better firefly.

At this stage, six better (Eq. 3) fireflies are selected to perform the operations based on their value function. In Takeuchi et al. (2015), the movements of male and female fireflies are modeled by physical differences. Male fireflies are attracted to all fireflies, while female fireflies are attracted to only male fireflies. Male fireflies move the same as fireflies of the OFA, and female fireflies move by new

position update method with adaptive control parameters (Cheng et al., 2023).

$$fitness = \text{minimize}(BW_i + CPU_i + \frac{1}{1+q_i}) \quad (3)$$

Where q is the firefly queue length, BW is the bandwidth, and CPU is the processor power.

Step 6: Modify the structure of the firefly.

$$\beta(r)_{male} = \beta_0 e^{-\gamma m}, m \geq 0 \quad (4)$$

$$\beta(r)_{female} = \beta_0 e^{-\gamma m} + 0.1 * \beta(r)_{male.Selected}, m \geq 0 \quad (5)$$

Here, r is the distance between two fireflies, β0 is the attraction at r=0 and γ is a constant light absorption coefficient. The distance between the female firefly and the nearest male firefly (d) is obtained by Eq. 6.

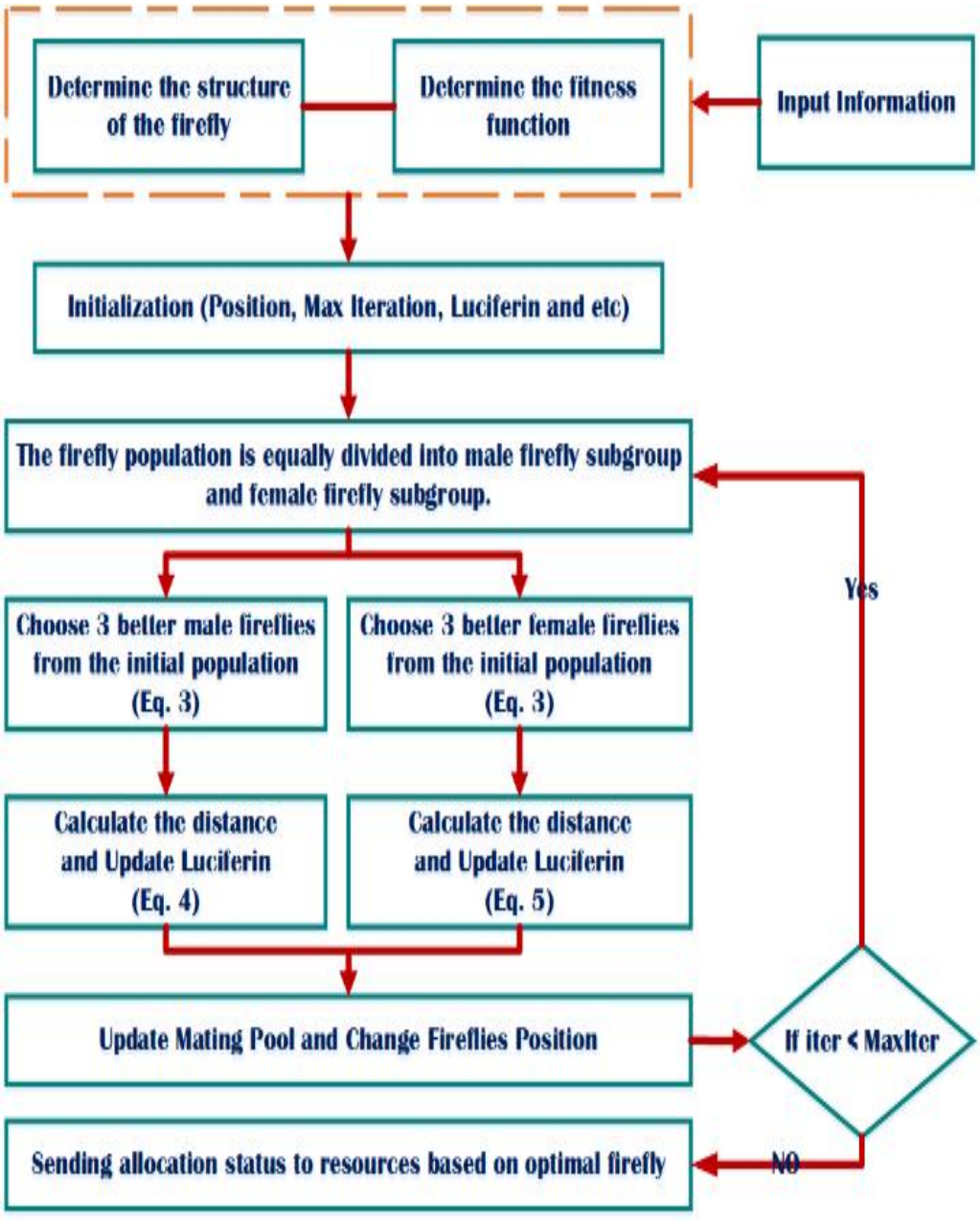


Fig. 3. Proposed research model.

At this stage, by changing Luciferin, spatial changes are obtained in proportion to other fireflies. This step gives the firefly algorithm the power to reach the optimal answer better and faster in different periods. A change in Luciferin changes the rate of firefly absorption. In the proposed model, the male firefly tends to other fireflies and the female firefly tends to the nearest male firefly. Eqs. 4 and 5 show the process of luciferin degradation in two types of fireflies.

$$d = \min(\sum \sqrt{(x_2 - x_1) * (y_2 - y_1)}) \quad (6)$$

Where x and y are the corresponding points of female firefly and male firefly.

Step 7: Select based on the load balance problem

At this stage, fireflies are selected that have been able to make better parameter changes in the way they move and place due to the load balance. Whenever a firefly changes its location-

like parameters, it must use those parameters to calculate the load balance. The worm changes are given to the load balance calculation formula, and the calculation is done. The better the balance, the higher the percentage of firefly selection.
How to select them is also in the form of a tournament so that all the parameters are applied to the load balancing formula, and then the answers that are better in the tournament will be selected.

Step 8: Check the end condition

The end conditions of the algorithm are twofold. The first is the number of repetitions set to 100 by default, and other cases have been considered. The second condition is to investigate whether fireflies have solved the load balancing problem on selected virtual machines. Changes may occur in each round, but the number of repetitions will not stop until the optimal load balance is achieved.

Step 9: Firefly changes

If the end conditions are not met, the firefighting location changes must be reapplied until selections are made. In each round, fireflies move to the optimal answer by changing their position.

4. Discussion and Experimentation

In this section, the proposed load-balancing algorithm is based on the firefly algorithm and its application in optimizing the processing of requests to the cloud network. The proposed approach is also about load balancing in the cloud system. The proposed approach is described in the various sections of Chapter Three. This section plans the experimental analysis of the proposed method by considering a cloud network through the MATLAB tool. Cloud simulation uses MATLAB programming like a program based on the core-i3 operating system, 8GB RAM, and 256 SSD.
In the simulation environment, a data center is defined. In this data center, it is possible to define any number of processes and virtual machines according to the user's preference. The data center can be defined as homogeneous or heterogeneous, and different processes can be defined, simulated, and executed on it. The proposed algorithm are studied in terms of parallelism parameters, scheduling of maximum process completion time, CPU efficiency, reducing the throughput of processors, and average time of processors in line of processors in different scenarios and it is compared with the results of these scenarios.

The evaluation is based on two general sections: "change in the initial parameters of the proposed algorithm" and "comparison of the proposed method with other algorithms." In the first part, the changes caused by each change are observed and recorded by applying changes to the firefly algorithm and virtual machines' initial parameters. In the second part the best results of these changes are compared with other methods that have worked on this issue.

4.1. Changes in the basic parameters of the proposed algorithm

This section evaluates the proposed method with other algorithms in task scheduling in cloud computing.

A) Change in the number of iterations of the program

In this section, the number of program iterations will change, but the other program parameters will be kept constant. The values of these parameters are given in Table 3. Fixed parameters are γ=1, β=2, and population size=100.

Table 3
Proposed algorithm parameters

Early population	Variable
γ	1, 1.5, 2
β	1, 1.5, 2
Max Iterations	100, 200, 500, 1000
Population Size	30, 50, 75, 100
Number of tasks	200, 400, 600, 800

According to the parameters, the program results for the number of iterations of 100, 200, 500, and 1000 are as follows.

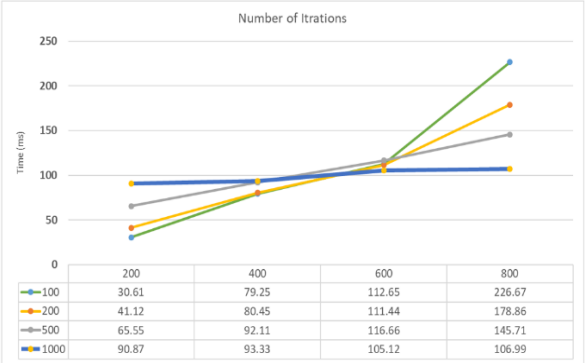


Fig. 4. Program results for different by of iterations.

As shown in the Figure 4, by changing the iteration of the program, the algorithm provides better answers. Figure 4 shows a comparison between different iterations. The constant trend of 1000 indicates the improvement of the situation in this number of iterations. Although iteration 1000 takes more time to balance in the case of 200 tasks, its steady trend for more tasks indicates that it is better.

B) Change in the initial population

In this section, the initial population will change, but the other program parameters are kept constant. The values of these parameters are shown in Table 3. Fixed parameters are γ=1, β=2, and iterations =1000.
As shown in the Figure 5, the algorithm provides better answers by changing the initial population. However, these answers produce far worse results than changing the number of repetitions of the program.

C) Change in γ and β values

In this section, the values of γ and β will change, but the other parameters of the program will be kept constant, given in Table 3 of these parameters' values. Fixed parameters are population size=100 and iterations =1000. According to the parameters, the program results for different values of γ and β are as follows.
As can be seen in the table above, by changing the values of γ and β, the algorithm offers different answers, which for the values of γ = 5.1 and β = 5.1, the proposed algorithm brings better answers to the output, in Figure 6, a comparison between different values of γ. Moreover, β is given.

Fig. 5. Graph changes based on initial population size.

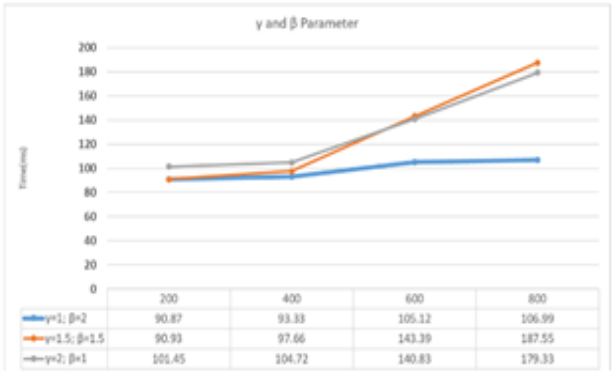


Fig. 6. Program results for different values of γ and β.

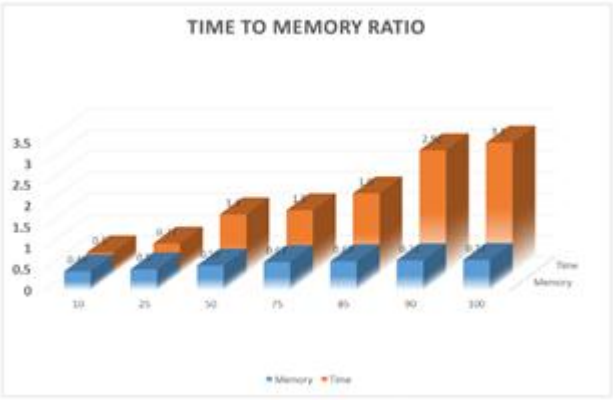


Fig. 7. Operating system software.

4.2. Comparison of proposed methods with other algorithms

In this section, the completion time of the last work of the proposed algorithm is compared with the standard genetic algorithm for different iterations.

A) Evaluation criteria

The main evaluation criteria used in the proposed cloud network scheduling method are effective for a schedule table's execution time. The main decision parameters to be considered include CPU usage and memory usage. The simulated datasets are compared based on different parameters to evaluate the performance of the proposed planning techniques. Time is also calculated based on the time required to produce an effective planning process.

B) Performance measurement

In performance evaluation, we consider the simulated cloud network as an evaluation system. The variable parameters in the proposed evaluation section are CPU interest rate and memory usage. Therefore, the mentioned decision parameters are considered as loads for the nodes. Therefore, the program production performance will be considered with the balance of given loads. In performance analysis, two types of analysis are presented.

Three different settings C1; C2; C3 are shown in Table 7 to obtain the desired output, and the simulator is set. The actual parameter settings are presented in Tables 4, 5, and 6.

Table 4
Cloud space configuration

Configuration	Number of users	Number of data centers
$C1$	25	10
$C2$	25	15
$C3$	30	20

Table 5
Basic physical settings

Memory	Storage space	The amount of memory available	Number of processors	Cpu speed
204800	100000000	1000000	4	10000

Table 6
Data center settings

Processor type	Operating system type	The cost of the virtual machine	Storage rate	Data center transfer rate
X64	Win2000	0.05	0.1	0.1

Table 7
User settings

Reply to user every hour	Data stored in each request	Starting rate by hour	Rate rates by hour
60	Win2000	0.05	0.1

B. 1) CPU interest rate analysis

In this analysis, the maximum amount of CPU usage in other credits is determined to evaluate the scheduling algorithm's performance. This analysis shows how the proposed approach performs well under different CPU load levels. Here we set the maximum CPU usage from 10 to 100. The results are shown in Figure 7. Timing and scheduling for different CPU rates are indicated by the timeline and memory used for synchronization. Memory usage is calculated in 100 KB. The analysis shows that the scheduling time is uniform for different CPU rates, and the memory also increases as the CPU rate increases.

B. 2) Analysis based on Memory Speed and CPU Rate

Here, the maximum amount of memory in other credits is set to evaluate the scheduling algorithm's performance. This analysis shows how the proposed approach shows high performance under different loads. The selected memory is set from 60 to 100, and the results of the analysis are

reviewed. The timeline and CPU rate line provide time and CPU for programming on different amounts of memory. CPU consumption rate is calculated at 100% CPU usage. The

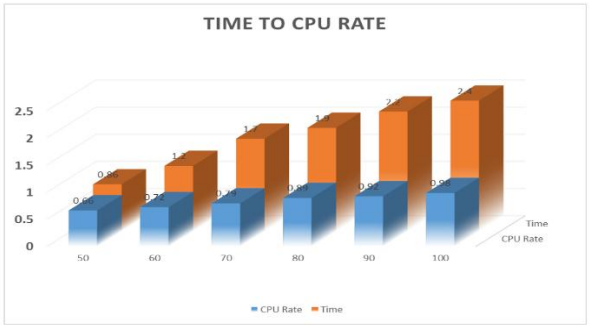


Fig. 8. Memory interest rate.

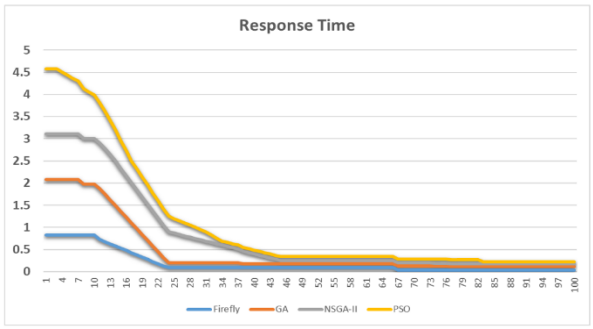


Fig. 9. Response time based on 21 users.

analysis shows that the time required to perform different tasks on different memory rates is uniform and balanced for CPU rates.

The defined cloud configuration setting provides different results than the initial state results, as shown in Figure 8. Sharing the load in the data center, provided in parallel, saves runtime for the request. Because the request is shared across different data centers, the number of requests per data center reduces processing time.

Figure 9 show the response time based on 21 users according to the C3 configuration on GA, NSGA-II, PSO, and Fireflies. As can be seen from the two diagrams, fireflies' response time is much better than others.

Figure 10 show the processing time for the 13 data centers on GA, NSGA-II, PSO, and Fireflies to the C3 configuration. Firefly processing time is much better than others.

B. 3) Comparative analysis

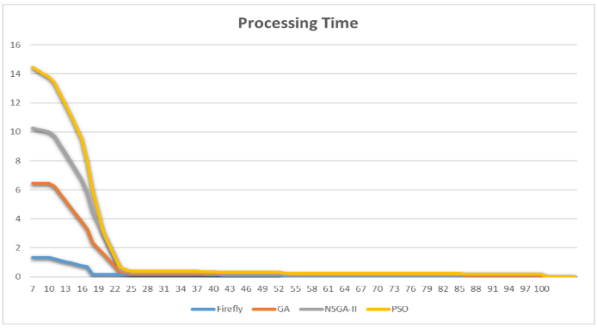
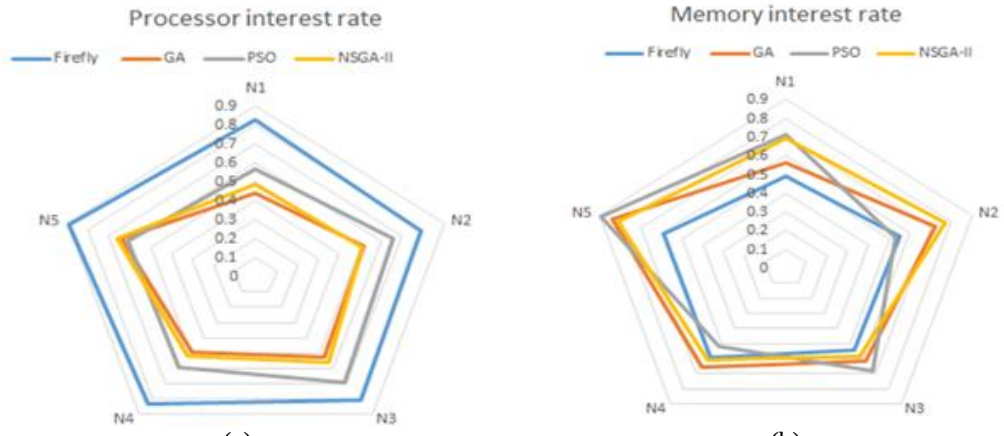


Fig. 10. Processing time for the 13 data centers.

The above section analyzes the performance of the proposed approach. To understand the importance of the proposed approach, we must compare it with other methods. Here, to prove the importance of evaluating the proposed approach to load balancing, the method was compared with GA, NSGA-II, and PSO. A comparative analysis of interest rates, CPU processing, and memory is presented. Figure 11-a show a comparative analysis of the proposed approach with the introduced approaches. The existing approach's values from the load-balanced scheduling methods obtained from the proposed method show that the proposed approach using a higher processing rate is more effective than the existing approach under load-balanced conditions. The analysis shows that the average amount of memory usage in the proposed approach is less. Therefore, considering the balanced load conditions and CPU interest rates, the proposed approach is more convenient than the existing approaches.

Figure 11-b shows a comparative analysis of the proposed approach with the introduced approaches. The existing approach's values show that the proposed approach using a lower execution time rate is more effective than the existing approach under load-balanced conditions. The analysis shows that the proposed method has reduced the load balancing time compared to the GA, NSGA-II, and PSO methods. According to the introduced diagrams, the proposed method has a much better performance than the introduced method.



(a) (b)

Fig. 11. Processor and Memory rate comparison chart.

5. Conclusions

The simulation results show that the proposed algorithm is highly dependent on the initial parameters. Among the initial parameters of the algorithm, the algorithm repetition rate yields much better results. The proposed method with 1000 repetitions gives us the best answer, which is the minimum makespan. Reducing makespan is effective in keeping virtual machines idle and balancing workload more in cloud environments. The proposed algorithm was also compared with other algorithms in the field of task scheduling. The results showed that the proposed algorithm has a much better execution time, completion time, and convergence process than other algorithms. This means that the proposed method is more effective and more efficient than the other two methods. Also, due to the vastness and dynamism of cloud computing data centers and the fact that these centers face a very diverse and variable workload, methods should be used to deploy virtual machines that are scalable, dynamic, and fast to be able to cope with such workload. These methods should behave optimally, and their efficiency should not decrease with increasing volume and variety of requests.
In future research, we plan to explore and assess different optimization algorithm models like Harris Hawks Optimization (HHO). Structural adjustments will be necessary in the firefly algorithm to accommodate the new algorithm.

References

Amiri, E., Roozbakhsh, Z., Amiri, S., & Asadi, M. H. (2020). Detection of topographic images of keratoconus disease using machine vision. International Journal of Engineering Science and Application, 4(4), 145-150.

Battula, A. R., & Vuddanti, S. (2022). Optimal reconfiguration of balanced and unbalanced distribution systems using firefly algorithm. International Journal of Emerging Electric Power Systems, 23(3), 317-328.

Bhoyar, A. A., & Dharmik, R. C. (2015). Design and implementation of job scheduling in grid environment over IPv6. IJCSMC, 4(4), 243-250.

Cardellini, V., Fanfarillo, A., & Filippone, S. (2017). Coarray-based load balancing on heterogeneous and many-core architectures. Parallel Computing, 68, 45-58.

Cheng, Z., Song, H., Zheng, D., Zhou, M., & Sun, K. (2023). Hybrid firefly algorithm with a new mechanism of gender distinguishing for global optimization. Expert Systems with Applications, 224, 120027.

Chung, I., & Bae, Y. (2004). The design of an efficient load balancing algorithm employing block design. Journal of Applied Mathematics and Computing, 14(1), 343-351.

Dam, S., Mandal, G., Dasgupta, K., & Dutta, P. (2014). An ant colony based load balancing strategy in cloud computing. In Advanced Computing, Networking and Informatics-Volume 2: Wireless Networks and Security Proceedings of the Second International Conference on Advanced Computing, Networking and Informatics (ICACNI-2014) (pp. 403-413). Springer International Publishing.

Dos Santos, M. J., & Fagotto, E. D. M. (2015). Cloud computing management using fuzzy logic. IEEE Latin America Transactions, 13(10), 3392-3397.

Gabhane, J. P., Pathak, S., & Thakare, N. M. (2023). A novel hybrid multi-resource load balancing approach using ant colony optimization with Tabu search for cloud computing. Innovations in Systems and Software Engineering, 19(1), 81-90.

Gerez, C., Silva, L. I., Belati, E. A., Sguarezi Filho, A. J., & Costa, E. C. (2019). Distribution network reconfiguration using selective firefly algorithm and a load flow analysis criterion for reducing the search space. IEEE Access, 7, 67874-67888.

Hasan, R. A., & Mohammed, M. N. (2017). A krill herd behaviour inspired load balancing of tasks in cloud computing. Studies in Informatics and Control, 26(4), 413-424.

Kalra, M., & Singh, S. (2015). A review of metaheuristic scheduling techniques in cloud computing. Egyptian informatics journal, 16(3), 275-295.

Kanbar, A. B., & Faraj, K. (2022). Region aware dynamic task scheduling and resource virtualization for load balancing in IoT–fog multi-cloud environment. Future Generation Computer Systems, 137, 70-86.

Kashikolaei, S. M. G., Hosseinabadi, A. A. R., Saemi, B., Shareh, M. B., Sangaiah, A. K., & Bian, G. B. (2020). An enhancement of task scheduling in cloud computing based on imperialist competitive algorithm and firefly algorithm. The Journal of Supercomputing, 76(8), 6302-6329.

Kaur, S., & Sengupta, J. (2017). Load balancing using improved genetic algorithm (iga) in cloud computing. Int. J. Adv. Res. Comput. Eng. Technol. (IJARCET), 6(8), 1323-2278.

Kazemi, A., Shiri, M. E., Sheikhahmadi, A., & Khodamoradi, M. (2021). A new parallel deep learning algorithm for breast cancer classification. International Journal of Nonlinear Analysis and Applications, 12(Special Issue), 1269-1282.

Kazemi, Z., Homayounfar, M., Fadaei, M., Soufi, M., & Salehzadeh, A. (2024). Multi-objective Optimization of Blood Supply Network Using the Meta-Heuristic Algorithms. Journal of Optimization in Industrial Engineering, 37(2), 63.

Keshk, A. E., El-Sisi, A. B., & Tawfeek, M. A. (2014). Cloud task scheduling for load balancing based on intelligent strategy. International Journal of Intelligent Systems and Applications, 6(5), 25.

Khaledian, N., & Mardukhi, F. (2022). CFMT: a collaborative filtering approach based on the nonnegative matrix factorization technique and trust relationships. Journal of Ambient Intelligence and Humanized Computing, 13(5), 2667-2683.

Khazaei, A., Haji Karimi, B., & Mozaffari, M. M. (2021). Optimizing the prediction model of stock price in pharmaceutical companies using multiple objective particle swarm optimization algorithm (MOPSO). Journal of Optimization in Industrial Engineering, 14(2), 73-81.

Lal, A., & Rama Krishna, C. (2018). Critical path-based ant colony optimization for scientific workflow scheduling in cloud computing under deadline constraint. In Ambient Communications and Computer Systems: RACCCS 2017 (pp. 447-461). Springer Singapore.

Li, J., Tian, Q., Zhang, G., Wu, W., Xue, D., Li, L., & Chen, L. (2018). Task scheduling algorithm based on fireworks algorithm. EURASIP Journal on Wireless Communications and Networking, 2018, 1-8.

Lotfi, M., & Behnamian, J. (2024). Virtual alliance in hospital network for operating room scheduling: Benders decomposition. Journal of Optimization in Industrial Engineering, 37(2), 15.

Mahendiran, A., Saravanan, N., Subramanian, N. V., & Sairam, N. (2012). Implementation of K-means clustering in cloud computing environment. Research journal of applied sciences, engineering and technology, 4(10), 1391-1394.

Maruthanayagam, D., & Prakasam, A. (2014). Job scheduling in cloud computing using ant colony optimization. Int. J. Adv. Res. Comput. Eng. Technol. (IJARCET), 3(2), 540-547.

Miao, Y. (2014). Resource scheduling simulation design of firefly algorithm based on chaos optimization in cloud computing. International Journal of Grid and Distributed Computing, 7(6), 221-228.

Navimipour, N. J., & Milani, F. S. (2015). Task scheduling in the cloud computing based on the cuckoo search algorithm. International Journal of Modeling and Optimization, 5(1), 44.

Negi, S., Rauthan, M. M. S., Vaisla, K. S., & Panwar, N. (2021). CMODLB: an efficient load balancing approach in cloud computing environment. The Journal of Supercomputing, 77(8), 8787-8839.

Raghava, N. S., & Singh, D. (2014). Comparative study on load balancing techniques in cloud computing. Open journal of mobile computing and cloud computing, 1(1), 18-25.

Rajeshkannan, R., & Aramudhan, M. (2016). Comparative study of load balancing algorithms in cloud computing environment. Indian Journal of Science and Technology, 9(20), 1-7.

Shadloo, N. (2017). A hybrid grey based two steps clustering and firefly algorithm for portfolio selection. Journal of Optimization in Industrial Engineering, 22(22), 49.

Sheikh, S., Nagaraju, A., & Shahid, M. (2021). A fault-tolerant hybrid resource allocation model for dynamic computational grid. Journal of Computational Science, 48, 101268.

Shokry, M., Awad, A. I., Abd-Ellah, M. K., & Khalaf, A. A. (2022). Systematic survey of advanced metering infrastructure security: Vulnerabilities, attacks, countermeasures, and future vision. Future Generation Computer Systems, 136, 358-377.

Takeuchi, M., Matsushita, H., Uwate, Y., & Nishio, Y. (2015). Firefly algorithm distinguishing between males and females for minimum optimization problems. submitted for publication.

Tapale, M. T., Goudar, R. H., Birje, M. N., & Patil, R. S. (2020). Utility based load balancing using firefly algorithm in cloud. Journal of Data, Information and Management, 2, 215-224.

Xu, B., & Sun, Z. (2016). A fuzzy operator based bat algorithm for cloud service composition. International Journal of Wireless and Mobile Computing, 11(1), 42-46.