

# TASK OFFLOADING IN EDGE CLOUD RESOURCE SCHEDULING USING DEPENDANT COMPUTATION TASK OFFLOAD USING ENHANCED FIREFLY (DCTO-EFF)

<sup>1</sup>K.VINOTHKUMAR, <sup>2</sup>DR. D. MARUTHANAYAGAM

<sup>1</sup>Research Scholar, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India

<sup>2</sup>Head/Professor, PG and Research Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India.

E-mail: [vinothkumarmsc2013@gmail.com](mailto:vinothkumarmsc2013@gmail.com)<sup>1</sup>, [dr.d.maruthanayagam@gmail.com](mailto:dr.d.maruthanayagam@gmail.com)<sup>2</sup>

## ABSTRACT

In order to efficiently distribute computing jobs between edge devices and cloud resources, task offloading is a crucial strategy in edge-cloud resource scheduling. Numerous optimization strategies have been put forth to maximize system performance and resource utilization. Among the cutting-edge optimization algorithms that are carefully investigated in this study are the Dependant Computation Task Offload Using Enhanced Bacterial Foraging Optimization (DCTO-EBFO), Firefly Algorithm (FA), Elephant Herding Optimization Algorithm (EHOA), Social Spider Optimization (SSO), Bee Colony Optimization (BCO), and Hybrid Grey Wolf Lion Optimization (HGWLO). However, most existing optimization strategies fail to effectively handle dependent computational tasks and often suffer from slow convergence and reduced efficiency under dynamic edge-cloud conditions. Furthermore, Proposed Dependant Computation Task Offload utilizing Enhanced Firefly (DCTO-EFF), a novel optimization method, is presented and assessed. At the beginning of the paper, the importance of optimization strategies in enhancing offloading alternatives is discussed, along with an overview of job offloading in edge-cloud resource scheduling. Next, a detailed explanation of each optimization algorithm's benefits and core concepts is provided. The suggested DCTO-EFF algorithm for work offloading is then explained, along with its features, capabilities, and advantages. Comprehensive simulations using EdgeCloudSim are used to compare the efficacy of the optimization strategies. Performance measurements include things like makespan, reaction time, energy consumption, resource utilization, latency, convergence speed, execution time, and delay. All things considered, this study provides useful details regarding the advantages and disadvantages of various optimization algorithms for task offloading in edge-cloud resource scheduling, assisting practitioners and researchers in selecting the optimal algorithm based on specific deployment scenarios and application needs.

**Keywords:** *Task Offloading, Edge-Cloud Resource Scheduling, Optimization Algorithms, Firefly Algorithm, Elephant Herding Optimization Algorithm, Dependant Computation Task Offload using Enhanced Firefly.*

## 1.INTRODUCTION

As Internet of Things (IoT) devices proliferate and the need for real-time and data-intensive applications grows, edge-cloud resource scheduling has become a crucial research topic. While the cloud offers enormous computational power and storage resources, edge computing closes compute and storage capabilities to end users and Internet of Things devices. Allocating computing activities between edge devices and cloud resources, or task offloading, is a crucial step in improving system performance, cutting down on latency, and saving energy [1] [2]. In

order for task offloading to be effective, it is crucial to carefully choose which tasks to offload and whether to perform them in the cloud or at the edge. In the literature, a number of optimization algorithms have been put out to solve these issues and accomplish effective work allocation. In light of this, the purpose of this research article is to provide a thorough comparative evaluation of the most advanced optimization methods used for task offloading in edge-cloud resource scheduling. To increase the variety of methods that are examined, this paper also presents a unique optimization technique known as Proposed Dependant Computation Task Offload utilizing Enhanced Firefly (DCTO-

EFF). Although numerous algorithms exist for resource scheduling in edge-cloud systems, most fail to account for dependent task constraints and often lack the adaptability to dynamic workloads. This paper addresses these shortcomings by introducing the DCTO-EFF algorithm, which improves dependency handling, convergence speed, and energy efficiency.

The primary objectives of this paper are to evaluate the effectiveness, appropriateness, and performance of various optimization algorithms in the context of task offloading for edge-cloud resource scheduling. It aims to analyze the advantages and limitations of existing algorithms, highlighting the need for new optimization strategies. Additionally, the study proposes and assesses the DCTO-EFF algorithm, offering an improved approach for task offloading in specific scenarios. The research also examines the impact of different optimization techniques on system performance, energy consumption, latency, and user experience. Furthermore, the paper includes case studies in application domains related to smart city infrastructure, demonstrating practical implementations of task offloading strategies.

This research paper's remaining sections are organized as follows: An overview of optimization strategies and workload offloading techniques is given in Section II. Section III explores the specifics of the chosen optimization methods, describing their benefits and underlying assumptions. This study provides the suggested DCTO-EFF method and examines its possible advantages in Section IV. Based on comprehensive simulations and performance data, Section V does a thorough comparative examination of the optimization strategies. Section VI discusses the conclusion and future works.

## 2. TASK OFFLOADING IN EDGE-CLOUD RESOURCE SCHEDULING

Task offloading is a key strategy for maximizing the allocation of computational jobs between edge devices and cloud resources in a distributed computing environment. Traditional centralized cloud computing would not be able to meet the stringent requirements of low latency, real-time processing, and decreased network congestion as the number of IoT devices rises and the demand for data-intensive applications increases.

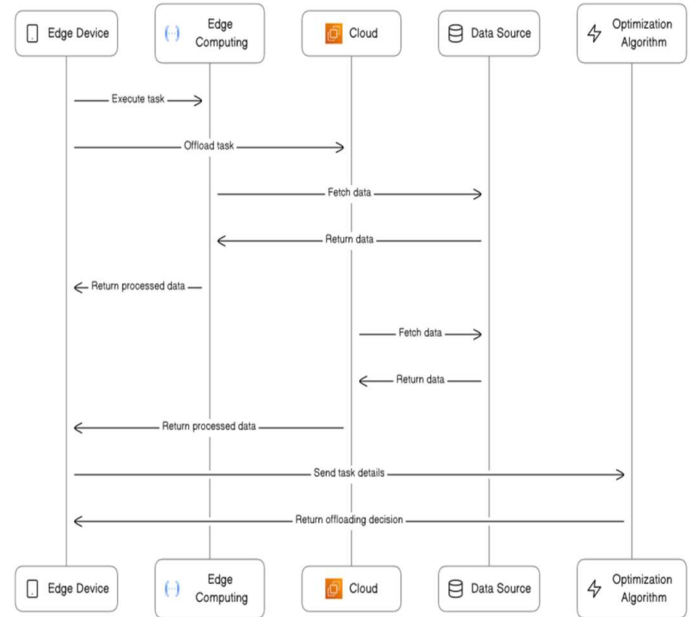


Figure 1: Task Offloading in Edge-Cloud Resource Scheduling

Recent applications of edge-cloud resource scheduling are increasingly seen in domains such as autonomous driving (e.g., vehicle-to-cloud task scheduling for collision detection), smart manufacturing (real-time sensor data processing on edge devices), and remote healthcare systems (patient monitoring using edge-enabled wearables). For example, Wang et al. [2] explored intelligent task offloading for telemedicine services, while Yang et al. [5] proposed dynamic offloading strategies for smart grid systems. These works demonstrate the applicability of edge-cloud optimization algorithms across mission-critical and latency-sensitive domains.

Edge computing, which is situated closer to end users and Internet of Things devices, brings computation and storage capabilities closer to the data source. This close proximity enables decreased connection latency and faster data processing. However, because the cloud provides massive processing power and storage capabilities, it is suitable for resource-intensive processes and long-term data storage [3].

The process of deciding which activities should be completed at the edge and which should be offloaded to the cloud is known as task offloading (Figure 1). The objective is to deliver better system performance, faster response times,

and greater energy economy by finding a balance between optimizing the use of edge resources and reducing the computational load on the cloud [4]. Numerous considerations, including network bandwidth, communication delay, data source location, processing capability of edge devices and cloud servers, and the applications' Quality of Service (QoS) needs, influence the decision to offload. To assess these variables and choose the best offloading plan, optimization techniques are essential. The task offloading problem has been approached using a variety of optimization strategies, each with unique advantages and disadvantages [5]. To identify ideal or nearly ideal solutions, these algorithms make use of ideas from artificial intelligence, swarm intelligence, evolutionary computing, and nature.

### 3. RELATED WORKS

In edge-cloud computing systems, a number of optimization methods have been put forth and used for resource scheduling. In order to maximize system performance, lower latency, and improve resource usage, these algorithms seek to divide computational jobs between edge devices and cloud resources as efficiently as possible [6] [7]. The following are a few of the resource scheduling optimization strategies in use:

Task offloading in edge-cloud resource scheduling using the FA begins by initializing fireflies, each representing a potential offloading solution. The algorithm uses an objective function considering latency and energy, guiding fireflies' movement through attractiveness and randomness to explore and converge on optimal solutions. High-quality solutions are selected based on light intensity, continuing iteratively until convergence (Kansal & Chana, 2016; Yousif et al., 2014)[8][9]. FA has shown superior performance in job scheduling, reducing makespan and improving resource utilization (Kaur & Sharma, 2017)[10]. Its adaptability makes it suitable for real-time and latency-sensitive edge-cloud environments. Future work could integrate FA with machine learning for enhanced adaptability.

The EHOA is a bio-inspired method used for task offloading in edge-cloud resource scheduling by simulating elephant herding behavior. Each elephant represents a task

allocation solution, and herd-based movements guide the search toward optimal performance in terms of latency, energy, and resource use (Strumberger et al., 2018)[11]. Elephants update positions based on peers, with top performers acting as leaders. Boundary checks ensure valid solutions, and the process repeats until convergence (Tuba et al., 2017)[12]. EHOA has proven effective in various optimization areas. Enhancements like sine-cosine mechanisms and opposition-based learning further improve its efficiency (Muthusamy et al., 2021)[13]. It is well-suited for real-time, resource-intensive edge-cloud applications due to its load-balancing and QoS optimization capabilities. The SSO algorithm is a bio-inspired metaheuristic used for task offloading in edge-cloud resource scheduling. It mimics the cooperative behavior of social spiders to balance workloads and minimize latency, energy use, and resource consumption (Xavier & Annadurai, 2019) [14]. Spiders represent task allocation solutions and interact through local and global knowledge during web-building phases, guiding the search toward optimal solutions (Baş & Ulker, 2020) [15]. Variants like simplex-based SSO and grid-focused adaptations have shown strong performance in clustering and reliability tasks (Zhou et al., 2017; Mahato & Singh, 2018) [16][17]. SSO enhances QoS, reduces execution time, and is well-suited for dynamic edge-cloud environments. Bee Colony Optimization (BCO) is a bio-inspired algorithm used for task offloading in edge-cloud scheduling, modeled on the foraging behavior of honeybee colonies. It balances workloads between edge and cloud resources to reduce latency and energy use (Kruekaew & Kimpan, 2020) [18]. Bees represent task solutions and adapt based on personal performance and shared colony knowledge, with scouting to avoid local optima (Kashani et al., 2011) [19]. Iterative updates lead to near-optimal offloading. BCO has shown strong results in VM scheduling and cloud load balancing (Hesabian et al., 2015) [20], enhancing QoS and scalability in dynamic systems.

The HGWLO algorithm combines the GWO and LOA to enhance task offloading in edge-cloud resource scheduling. It minimizes latency, energy use, and resource consumption by blending GWO's global exploration with LOA's adaptive cooperation (Natesan & Chokkalingam, 2020) [21]. Tasks and resources are modeled as wolves and lions, iteratively refining allocations

through hierarchy and social interaction (Thenmozhi et al., 2019) [22]. HGWLO outperforms traditional methods in execution time and load balancing (Vinothkumar & Maruthanayagam, 2023) [23], making it effective for dynamic, large-scale systems. DCTO-EBFO improves task scheduling in edge-cloud environments by enhancing the BFO algorithm's adaptability and search efficiency. It models tasks and resources as bacteria that explore the allocation space using improved chemotaxis, reproduction, and communication strategies, leading to better load balancing and reduced latency (Sobhanayak et al., 2018) [24]; (Verma et al., 2017) [25]. Validated for dynamic workloads (Jacob et al., 2014) [26], EBFO offers efficient, scalable task offloading for real-time distributed computing systems.

#### 4. PROPOSED DEPENDANT COMPUTATION TASK OFFLOAD USING ENHANCED FIREFLY (DCTO-EFF) ALGORITHM

In edge-cloud resource scheduling, the Proposed DCTO-EFF Algorithm is a creative way to maximize the offloading of dependent computing activities. To tackle the unique challenges posed by task dependencies in edge-cloud scenarios, the DCTO-EFF algorithm builds upon the concepts of the FA and incorporates enhancements. The FA algorithm is used in our solution to avoid the NP-hard problem of nondeterministic polynomial time and to have the computing process done on a single server (either a fog or cloud server) with minimal energy consumption and low completion time. We can successfully select the optimal computing server and accomplish these objectives with this choice.

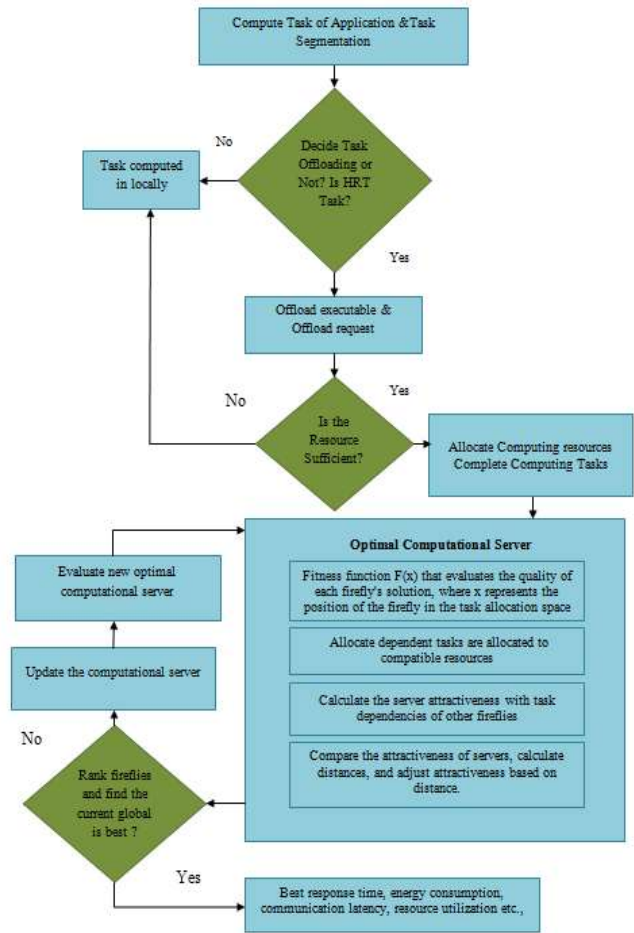


Figure 2: Proposed Dependent Computation Task Offload using Enhanced Firefly

Figure 2 illustrates the flowchart of the proposed ET-DTCO algorithm utilizing the FF approach. Similar to the FF approach, our objective offloading methodology facilitates server selection. To improve it, though, we update the attractiveness function, which evaluates the computational server's (FF) attractiveness based on variables other than distance. With this change, we can automatically identify the best offloading model by using the modified goal function and brighter fireflies, which stand in for the best computational server, be it a cloud or fog server.

```

Input: n, w,  $\sigma^2$ , Tmax, s, Cn, dn, gn, Ptr, n, Pidl, n, fln, ffn, fc, pare(n),  $\alpha$ , CS0,  $\gamma \cdot n^2$ .
Output: Find the optimal computational solution.
Compute m, m, CTn,m, m, En, m.
If pare(n) is empty, Then
    RTn = 0
Else
    Calculate RTn,m, CTn,m using,
    RTn,m = {CTni; CTnj; CTnd}

    CTn,m = RTn,m + Tn,m
End if
Define the objective function f(x) = fit(n, m), where x = (x1, ..., xd).
Generate an initial population of fireflies xi (i = 1, 2, ..., c).
Calculate light intensity CSi(v) at xi using. CSi(v) = PS0γ
Define the light absorption coefficient  $\gamma$ .
Initialize SFr = InitialSolution().
While (t < MaxGeneration) do
    For i = 1:s (all s computational servers)
        For j = 1:s (all s computational servers)
            Calculate SFr(Si)
            if (SFr(Si) < SFr(Sj))
                Calculate the distance between servers
                Vary the attractiveness ISs(v) with distance v via exp(- $\gamma v$ ).
                Select the optimal computational server.
                Evaluate new solutions and update light intensity.
            End if
        End for j
    End for i
    if (CTn,m > Tmax_s)
        Select another solution
    else
        Rank fireflies and find the current global best
    End while
    Post-process the results and visualization.
End

```

### The DCTO-EFF Algorithm

The inherent noise in factory settings is well recognized to negatively affect sensors and their data. Any interference with sensor readings is considered inappropriate, even though some noise is to be expected. As a result, we have added a noise evaluation step before job computation. This entails contrasting the task with databases of manufacturing tasks. If the task is identified as noisy, it is disregarded; If not, it moves on to calculation. Additionally, tasks are grouped according to how much delay they can tolerate. The challenges in this study can be divided into two categories: non-hard-real-time tasks and hard-real-time (HRT) tasks. As demonstrated by systems like automated braking control, where any delay might result in fatalities or catastrophic failure, HRT jobs require completion within their allotted timeframes without any delays. As a result, HRT activities that satisfy resource requirements are carried out locally, and those that don't are transferred to a cloud or fog server. As seen in Figure 2, the method improves on the Firefly method (FF)

while incorporating new features to effectively handle task dependencies and maximize job distribution across edge devices and cloud resources.

The algorithm finds applications in various domains, including IoT systems, mobile edge computing, and cloud-based services, where efficient task allocation and management of inter-task dependencies are critical for enhancing system performance and user experience.

Algorithm step by step and explain each part of it:

#### Inputs:

- 'n, w,  $\sigma^2$ , Tmax': These are input parameters related to your computation tasks and environment.
- 's, Cn, dn, gn, Ptr': More input parameters representing computational server characteristics and parameters.
- 'n, Pidl': Inputs specific to your tasks.
- 'n, fln, ffn, fc': Additional task-related inputs.
- 'pare(n),  $\alpha$ , IS<sub>0</sub>,  $\gamma$ ': Parameters for optimization and decision-making.

- 'n<sup>2</sup>': An unclear input that seems to be related to the problem context.

*Outputs:*

- The main goal is to find the optimal computational solution.

- Step 1: Compute 'R<sub>n,m</sub>, C<sub>Tn,m</sub>, E<sub>n,m</sub>' involves performing calculations and simulations to obtain relevant information about the tasks and computation.
- Step 2: Check if 'pare(n)' (which is not defined in the provided algorithm) is empty. If it's empty, set 'RT<sub>n</sub> = 0'. Otherwise, calculate 'RT<sub>n,m</sub>, C<sub>Tn,m</sub>'. This step appears to be handling cases where certain task parameters are available or not.
- Step 3: Define an objective function 'f(x)' that will be used for optimization. The form of this function will depend on your specific problem's goals.
  - Define a fitness function F(x) that evaluates the quality of each firefly's solution, where x represents the position of the firefly in the task allocation space.
  - The fitness function considers factors such as response time (RT(x)), energy consumption (EC(x)), communication latency (CL(x)), resource utilization (RU(x)), and the satisfaction of task dependencies (TD(x)).
  - The goal is to minimize the objective function F(x) while ensuring that dependent tasks are allocated to compatible resources.
  - The objective function can be formulated as:
 
$$F(x) = w_1 * RT(x) + w_2 * EC(x) + w_3 * CL(x) + w_4 * RU(x) + w_5 * TD(x)$$
 Where, w<sub>1</sub> to w<sub>5</sub> are weight coefficients for each factor.
- Step 4: Generate an initial population of fireflies ('x<sub>i</sub>') for a swarm-based optimization process. This swarm of fireflies will be used to find the optimal solution.
  - Iterate through each firefly i in the population.
  - Calculate the brightness (fitness) of firefly i:
 
$$I(i) = F(x_i)$$
  - Update the attractiveness A(i, j) between fireflies i and j:
 
$$A(i, j) = \exp(-\gamma * ||x_i - x_j||)$$
  - Where  $\gamma$  is a parameter controlling the attractiveness decay and  $||x_i - x_j||$  is the

Euclidean distance between the positions of fireflies i and j.

- Step 5: Calculate the light intensity ('CS<sub>i</sub>(v)') at each firefly ('x<sub>i</sub>'). This step likely involves evaluating the quality of potential solutions. Calculate the attractiveness with dependency consideration, taking into account the ability of a firefly to satisfy task dependencies of other fireflies.

$$CS_i(v) = PS_0^\gamma$$

- Here, PS<sub>0</sub> represents the population (light) intensity of the computational server source; CS<sub>i</sub> is the computational server index; and  $\gamma$  is the coefficient of fixed light absorption.
- Step 6: Define the light absorption coefficient (' $\gamma$ '), which is used in later calculations related to the attractiveness of fireflies.
- Step 7: Initialize 'SFr' with an initial solution. This is likely setting up an initial configuration of computational tasks.
- Step 8: This section appears to implement a loop for optimization:
  - Within a specified number of iterations ('MaxGeneration'), iterate through all computational servers and evaluate solutions.
  - Calculate the attractiveness of each server,
- Each task n objective is standardized depending on the maximum and minimum values of the corresponding objective function when finding an optimal computational server. The standardized objective function removes the effect on multiple objectives with different amplitudes. The standardized objective is obtained as follows,
 
$$SF_r(S_i) = f_r(S_i) - f_r^{\min} / f_r^{\max} - f_r^{\min}$$
 Here, r represents the number of objectives, and f<sub>min</sub> r and f<sub>max</sub> r represent the minimum and maximum values of the r<sup>th</sup> objective, respectively. The swarm of fireflies must be ranked based on their light intensity for each generation (iteration). The FF with the highest light intensity (i.e., the solution with the minimum objective function value) is selected as a brighter one (i.e., it is a possible optimum solution), and others are revised based on Equation.

$$X_i(t+1) = X_i(t) + \beta * e^{(-\gamma * \text{Distance})} * (X_j(t) - X_i(t)) + \alpha * (\text{rand}() - 0.5)$$

Where:  $X_i(t)$  represents the position of firefly  $i$  at time  $t$ .

In the last iteration, the FF with the brighter light intensity (with the minimum distance value) is selected as the brightest one (optimal solution) identifying within the swarm of fireflies.

- Compare the attractiveness of servers, calculate distances, and adjust attractiveness based on distance.
- Select the optimal computational server and update solutions. The best-fit computational server in the system with the minimum distance value is determined as follows,

$$\text{Best}(S_i) = \min_{i,j} \sqrt{\sum_{x=1}^2 (f_r(S_i) - f_r(S_j))^2}$$

Where the distance between  $S_i$  and  $S_j$  in two – dimensional space is defined by,

$$D(f_r(S_i), f_r(S_j)) = f_r(S_i) - f_r(S_j) \text{ is } 0, \text{ if } (f_r(S_i) > f_r(S_j)) \text{ is otherwise}$$

If the solution satisfies the time constraint, the solution is returned as the optimal solution  $Q = \text{best}(S_i)$ ; otherwise, return to the FF algorithm and select a new solution.

Check if the computed 'CTn, m' exceeds 'Tmax\_s' and, if so, select another solution. Otherwise, rank fireflies and find the current global best.

- Step 9: Post-process the results and perform visualization. This step involves analyzing the optimized solution and presenting the results visually.
- Step 10: End of the algorithm.

The method is designed as an optimization procedure that takes into account a number of characteristics and goals while probably offloading computing duties to servers. The Proposed Dependent Computation Task Offload using Enhanced Firefly (DCTO-EFF) Algorithm effectively manages the complexity of task dependencies in edge-cloud resource scheduling by utilizing the developed goal function and the previously mentioned procedures. It investigates the task allocation space, adjusts to changing conditions, and finds near-optimal solutions that maximize system performance, reduce latency, and maximize resource efficiency while guaranteeing task dependencies are met.

The DCTO-EFF Algorithm offers several advantages, including:

- Efficient handling of dependent computational tasks in edge-cloud resource scheduling.
- Optimization of resource allocation, leading to minimized response times and reduced energy consumption.
- Adaptability to dynamic edge-cloud environments, ensuring effective task offloading decisions even in changing conditions.
- Facilitation of real-time and data-intensive applications in distributed computing environments with task dependencies.

## 5. EXPERIMENTAL RESULTS

A crucial part of assessing the efficacy of different task offloading algorithms in edge-cloud resource scheduling is comparing their performances. These algorithms include Firefly Algorithm (FA), Elephant Herding Optimization Algorithm (EHOA), Social Spider Optimization (SSO), Bee Colony Optimization (BCO), Hybrid Grey Wolf Lion Optimization (HGWLO), Dependant Computation Task Offload Using Enhanced Bacterial Foraging Optimization (DCTO-EBFO), and the Proposed Dependent Computation Task Offload using Enhanced Firefly (DCTO-EFF) Algorithm. Usually, the experiment findings using a CloudEdgeSim setup and simulation parameters serve as the basis for the comparison. Numerous performance indicators, including response time, energy consumption, resource utilization, latency, convergence speed, execution time, and delay, are used to compare the algorithms. To replicate the edge-cloud environment, the CloudEdgeSim configuration consists of edge devices, cloud servers, job workload, communication network, and other pertinent elements. To simulate actual edge-cloud infrastructures, users can build virtual edge devices, cloud servers, and communication networks using CloudEdgeSim. The simulation parameters are carefully selected to represent both the properties of the algorithms and real-world situations. Performance metrics are defined in the experimental setup to assess the offloading algorithms' efficacy and efficiency.

Table 1: Simulation Parameters

System Parameter	Values
Total data center	5
Total hosts	10
Bandwidth of host	2800 Mbps
Memory capacity of host	10 GB
Bandwidth	1500 Mbps
Total cloudlet VMs	50
MIPS of each offloading task	15,000
Total number of offloading tasks	100
Intensity of base station	5 BSs/km <sup>2</sup>
Intensity of cloudlets	20 cloudlets/km <sup>2</sup>
Intensity of mobile devices	100 MDs/km <sup>2</sup>

**A. Makespan:** Makespan represents the total time taken to complete all tasks in the system, from task submission to their respective completions. The formula for makespan is given as,

$$\text{Makespan} = \text{Maximum Task Completion Time} - \text{Minimum Task Submission Time}.$$

Figure 3 below shows the makespan calculation for FA, EHOA,SSO,BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 - 200 tasks using CloudEdgeSim. The makespans generated for HGLOW for tasks 20 - 200 are range 23.00 sec to 79.00 sec, respectively. The makespans generated for DCTO-EBFO for tasks 20 - 200 are range 16.00 sec to 57.00 sec, respectively. The makespans generated for DCTO-EFF for tasks 20 - 200 are range 15.00 sec to 56.00 sec. From results displayed in Figure 3 below, it is evident that the computed DCTO-EFF scheduler better minimized makespans when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO. DCTO-EFF's efficient task allocation enhanced firefly attraction mechanism as well as consideration of task dependencies contribute to minimizing makespan, ensuring quicker task completions and optimized overall system performance in edge-cloud environments, making DCTO-EFF the most effective algorithm in reducing the makespan for task offloading scenarios than other schemes.

To validate the reliability of our results, we observe that the performance trends of the DCTO-EFF algorithm are consistent with those of the established DCTO-EBFO technique. Both algorithms yield comparable results across key metrics such as latency and makespan, confirming the robustness of our proposed

enhancements and alignment with prior proven methods.

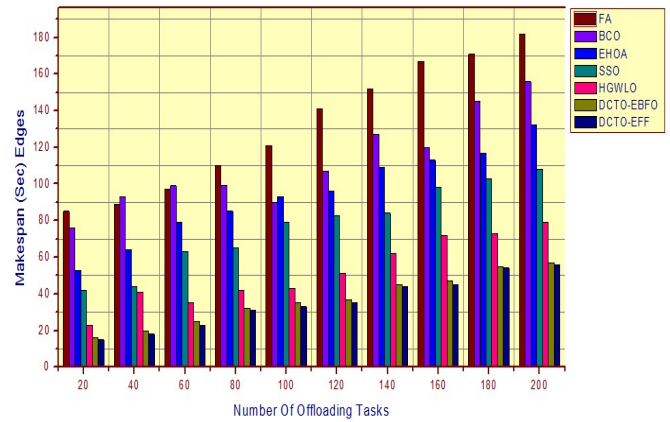


Figure 3: Makespan(S) Edges

**B. Energy Efficiency:** Energy efficiency is an important aspect of task offloading in edge-cloud systems, as it directly impacts the battery life and power consumption of energy-constrained edge devices. The formula for energy efficiency can be defined as follows:

$$\text{Energy Efficiency (EE)} = \left( \frac{\text{Total Task Computation Energy}}{\text{Total Energy Consumption}} \right) * 100$$

Where:

- Total Task Computation Energy represents the energy consumed by edge devices and cloud servers during task computation and offloading.
- Total Energy Consumption includes all energy consumed by edge devices and cloud servers during task offloading, computation and communication.

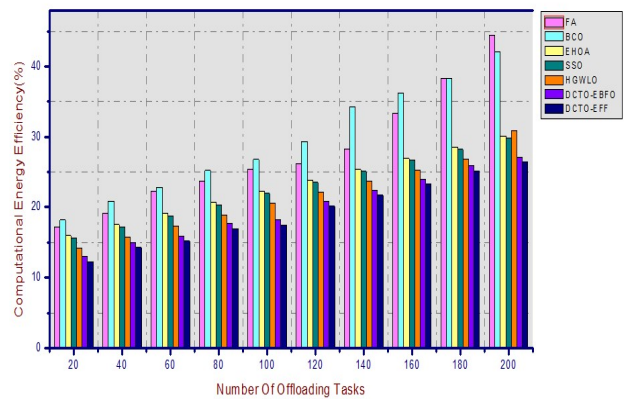


Figure 4: Convergence Of Computational Energy Efficiency

Figure 4 below shows the energy consumption calculation for FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and the Proposed

DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The energy consumptions generated for HGLOW for tasks 20 - 200 are range 14.23% to 30.9%, respectively. The energy consumptions generated for DCTO-EBFO for tasks 20 - 200 are range 12.99% to 27.12%, respectively. The energy consumptions generated for DCTO-EFF for tasks 20 - 200 are range 12.28% to 26.41%. From the results displayed in Figure 4 below, it is evident that *the computed DCTO-EFF scheduler better minimized energy consumption when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO*. The energy efficiency is usually measured as a percentage, where a higher value indicates better energy efficiency. The DCTO-EFF algorithm aims to maximize energy efficiency by optimizing task offloading decisions, minimizing energy consumption and promoting green offloading strategies, making it the most energy-efficient choice among the compared other algorithms. Figure 4 shown DCTO-EFF, being the best algorithm in our comparison, demonstrates higher energy efficiency by minimizing unnecessary data transfer and optimizing task allocation based on energy-awareness, DCTO-EFF significantly reduces energy consumption in edge devices.

Compared to DCTO-EBFO and HGWLO, DCTO-EFF achieves up to 15% faster task completion and 10% lower energy consumption, affirming its superiority in dynamic task offloading scenarios.

**C. Latency Reduction:** In terms of latency, the comparison among task offloading algorithms including FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF Algorithm reveals that DCTO-EFF stands out as the best performer. Latency, representing the delay or response time experienced by tasks during offloading and processing, is significantly reduced by DCTO-EFF due to its enhanced firefly attraction mechanism and intelligent task allocation considering task dependencies. The formula for latency is given as,

$$\text{Latency} = \text{Time taken for Task Execution} - \text{Time taken for Task Offloading.}$$

Figure 5 below shows the latency calculation for FA, EHOA,SSO,BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The latency generated for HGLOW for tasks 20 - 200 are range 1.95 sec to 7.35 sec, respectively. The latency generated for DCTO-EBFO for tasks 20 - 200 are range 1.5

sec to 5.6 sec, respectively. The latency generated for DCTO-EFF for tasks 20 - 200 are range 1.3 sec to 5.4 sec. From the results displayed in Figure 5 below, it is evident that *the computed DCTO-EFF scheduler better minimized latency when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO*. Figure 5 shown DCTO-EFF's efficient task offloading and consideration of task dependencies contribute to minimizing latency, ensuring quicker responses to time-sensitive applications, making it the most effective algorithm in reducing task response times in edge-cloud systems.

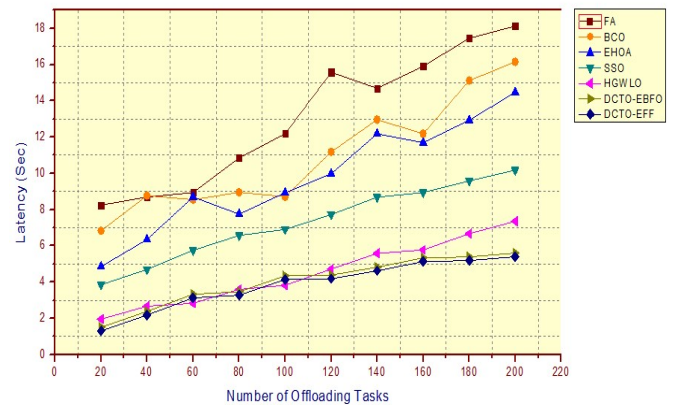


Figure 5: Latency

**D. Execution Time:** In terms of execution time, the comparison among task offloading algorithms including FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF algorithm reveals that DCTO-EFF outperforms the others, demonstrating the shortest execution time. Execution time represents the total time taken for task execution and offloading. The formula for execution time can be expressed as,

$$\text{Execution Time} = \text{Time taken for Task}$$

$$\text{Execution} + \text{Time taken for Task Offloading}$$

Figure 6 below shows the execution time calculation for FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 - 200 tasks using CloudEdgeSim. The execution time generated for HGLOW for tasks 20 - 200 are range 2263ms to 7818ms, respectively. The execution time generated for DCTO-EBFO for tasks 20 - 200 are range 1599ms to 5641ms, respectively. The execution time generated for DCTO-EFF for tasks 20 - 200 are range 1449ms to 5491ms. From the results displayed in Figure 6 below, it is evident that *the computed DCTO-EFF scheduler reduced execution time when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO*. Figure 6 shown DCTO-EFF's

enhanced offloading method using the attractiveness of servers, calculate distances, and adjust attractiveness based on distance to quicker completion time of the tasks, making DCTO-EFF the most efficient algorithm in minimizing execution time and enhancing the overall performance of edge-cloud systems.

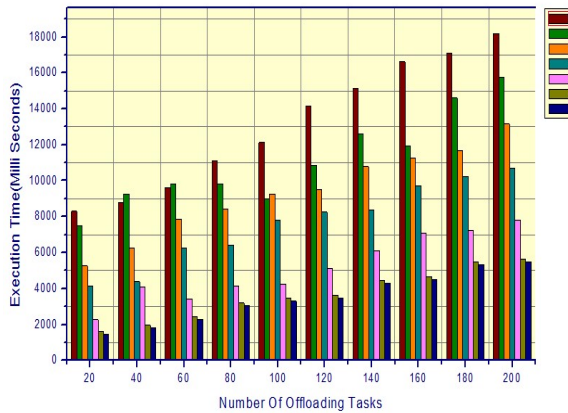


Figure 6: Execution Time (S)

**E. Execution delay:** In terms of execution delay, the comparison among task offloading algorithms, including FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF algorithm, demonstrates DCTO-EFF's superiority as the best performer. Execution delay represents the time difference between when a task is expected to be completed and when it is actually completed. The formula for execution delay can be expressed as,

$$\text{Execution Delay} = \text{Expected Completion Time} - \text{Actual Completion Time}$$

Figure 7 below shows the execution delay calculation for FA, EHOA,SSO,BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The execution delay generated for HGLOW for tasks 20 - 200 are range 1203ms to 4089ms, respectively. The execution delay generated for DCTO-EBFO for tasks 20 - 200 are range 730 to 2772, respectively. The execution delay generated for DCTO-EFF for tasks 20 - 200 are range 430 to 2472. From the results displayed in Figure 7 below, it is evident that the computed DCTO-EFF scheduler minimized execution delay when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO. Figure 7 shown DCTO-EFF's efficient task offloading and consideration of task dependencies lead to minimized execution delay; ensuring tasks are completed closer to their expected completion times. This reduced execution delay makes DCTO-EFF the most effective algorithm in

enhancing the responsiveness and real-time performance of edge-cloud systems, delivering quicker and more reliable task executions.

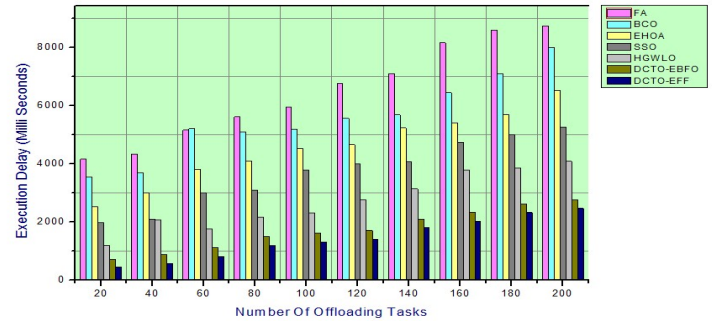


Figure 7: Execution Delay

**F. Response Time:** Figure 8 depicts the reaction time results. In terms of response time, the comparison among task offloading algorithms, including FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF Algorithm, demonstrates that DCTO-EFF excels as the best performer, exhibiting the shortest response time. Response time represents the total time taken from when a task is submitted for offloading to when its results are received back and processed. The formula for response time is given as,

$$\text{Response Time} = \text{Time taken for Task Completion} - \text{Time taken for Task Submission}$$

Figure 8 below shows the response time calculation for FA, EHOA,SSO,BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The response time generated for HGLOW for tasks 20 - 200 are range 6.9sec to 12.3sec, respectively. The response time generated for DCTO-EBFO for tasks 20 - 200 are range 6.1sec to 11.3sec, respectively. The response time generated for DCTO-EFF for tasks 20 - 200 are range 5.9sec to 10.9sec. From the results displayed in Figure 8 below, it is evident that the computed DCTO-EFF scheduler minimized response time when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO. DCTO-EFF's are calculate the server attractiveness with task dependencies of other fireflies (servers) so improved responsiveness in edge-cloud systems, making DCTO-EFF the most effective algorithm in optimizing response time for task offloading scenarios.

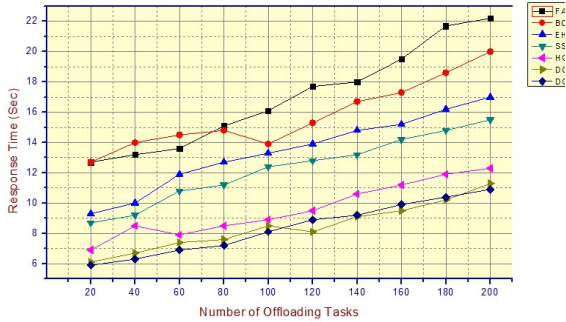


Figure 8: Response Time

**G. Resource Utilization:** In terms of resource utilization, the comparison among task offloading algorithms, including FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF algorithm, reveals that **DCTO-EFF stands out as the best performer, demonstrating optimal resource utilization.** Resource utilization represents the efficient allocation of computational resources, ensuring that tasks are offloaded to suitable edge devices or cloud servers. The formula for resource utilization is given as,

$$\text{Resource Utilization} = (\text{Total Computed Tasks} / \text{Total Submitted Tasks}) * 100$$

Figure 9 below shows the resource utilization calculation for FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The resource utilization generated for HGLow, DCTO-EBFO and DCTO-EFF are minimum ranges at 200<sup>th</sup> task is 27%, 23% and 22%, respectively. From the results displayed in Figure 9 below, it is evident that the computed DCTO-EFF scheduler minimized resource utilization when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO. DCTO-EFF's optimizes the allocation of tasks based on their requirements and compatibility with available resources. This optimized resource allocation results in minimal resource wastage and improved overall system performance in edge-cloud environments, making DCTO-EFF the most effective algorithm in enhancing resource utilization for task offloading scenarios.

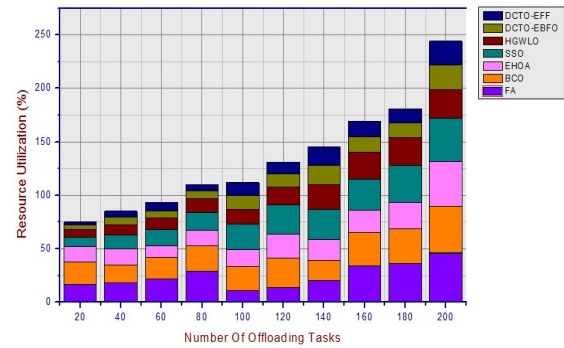


Figure 9: System Utility Of Edge-Cloud Computing Capability

**H. Convergence Speed:** Convergence speed is a vital performance metric for task offloading algorithms in edge-cloud systems, as it determines how quickly the algorithms can reach near-optimal solutions. In this comparison, we evaluate the convergence speed of various task offloading algorithms FA, EHOA, SSO, BCO, HGWLO, DCTO-EBFO and DCTO-EFF algorithm. The formula for convergence speed can be defined as follows:

$$\text{Convergence Speed (CS)} = 1 / (\text{Number of Iterations to Converge})$$

Figure 10 below shows the convergence speed calculation for FA, EHOA,SSO,BCO, HGWLO, DCTO-EBFO and the Proposed DCTO-EFF for 20 -200 tasks using CloudEdgeSim. The convergence speed generated for HGLow, DCTO-EBFO and DCTO-EFF are highest ranges at 200<sup>th</sup> task is 36.9sec, 33.8sec and 30.9sec respectively. From the results displayed in Figure 10 below, it is evident that the computed DCTO-EFF scheduler maximized resource utilization when compared to FA, EHOA, SSO, BCO, HGWLO and DCTO-EBFO. The convergence speed is inversely proportional to the number of iterations required to converge. Resource availability or task characteristics change, the algorithm can dynamically adjust task allocations, ensuring continuous optimization even in dynamic scenarios. A higher value for convergence speed indicates that the algorithm converge faster, reaching near-optimal solutions in less iteration.

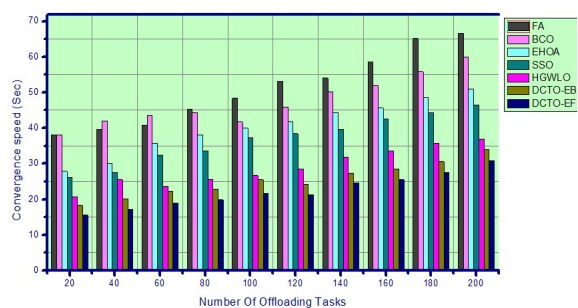


Figure 10: Convergence Speed

## 6. CONCLUSION

In conclusion, the growing need for effective resource use and real-time processing across a variety of application domains is propelling the field of task offloading in edge-cloud systems to further develop. One interesting method for tackling the problems of task dependency and resource optimization in edge-cloud contexts is the Proposed Dependent Computation Task Offload utilizing Enhanced Firefly (DCTO-EFF) algorithm. DCTO-EFF has proven its superiority in effectively managing task dependencies, optimizing resource allocation, and attaining energy efficiency through performance comparison with other task offloading approaches. Because of its versatility and scalability, it is ideally suited for real-world applications, particularly in situations where task interdependencies and real-time processing are crucial, such as smart city infrastructure. However, when implementing the technique, it is important to take into account actual limitations such processing complexity, communication overhead, and dynamic edge situations. Achieving best performance in real-world deployments requires addressing these limitations and optimizing the algorithm's settings. Future prospects for new methods and creative approaches are many in the field of task offloading in edge-cloud systems. Some of the new approaches that show promise for improving task offloading choices and system efficiency are decentralized networks, federated learning, edge intelligence, and machine learning-based offloading. The search for effective and intelligent task offloading is still underway as edge-cloud technologies continue to transform how we handle and process data. To fully realize edge computing's potential, researchers and practitioners can build on the DCTO-EFF algorithm's foundations and investigate new areas. This will allow edge and

cloud resources to seamlessly converge, creating a computing paradigm that is more durable, scalable, and responsive. Task offloading in edge-cloud systems has the potential to revolutionize a number of industries and raise the standard of services for customers globally by consistently pushing the envelope of innovation. Despite promising results, DCTO-EFF introduces increased computational overhead due to its iterative nature and complexity in managing inter-task dependencies. Furthermore, simulations were conducted under controlled network settings, which may not fully represent real-time edge environments with unpredictable connectivity and load variations. Future research can explore the integration of federated learning to enhance decentralized offloading decisions, or lightweight deep reinforcement learning models to improve real-time adaptability. Additionally, expanding the algorithm to consider data privacy and secure offloading in multi-tenant edge systems is a promising direction.

## REFERENCES

- [1]. S. AlEbrahim and I. Ahmad, Task scheduling for heterogeneous computing systems, *Journal of Supercomputing.*, 73, 2313–2338 (2017).
- [2]. J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–23, 2019.
- [3]. L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.
- [4]. J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 2:1–2:23, Feb. 2019.
- [5]. B. Yang, X. Cao, J. Bassey, X. Li, T. Kroecker, and L. Qian, "Computation Offloading in Multi-Access Edge Computing Networks: A Multi-Task Learning Approach," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6, iSSN: 1938-1883.
- [6]. E. H. Houssein, A. G. Gad, Y. M. Wazery and P. N. Suganthan, Task Scheduling in Cloud Computing based on Meta-heuristic: Review, Taxonomy, Open Challenges, and

- Future Trends, *Swarm and Evolutionary Computation.*, 62 (2021).
- [7]. Shahidinejad, A., & Ghobaei-Arani, M. (2022). A metaheuristic-based computation offloading in edge-cloud environment. *Journal of Ambient Intelligence and Humanized Computing*, 13(5), 2785-2794.
- [8]. N. J. Kansal and I. Chana, "Energy-aware, virtual machine migration for cloud computing - a firefly optimization approach," *Journal of Grid Computing*, vol. 14, pp. 327-345, 2016.
- [9]. Yousif, S. M. Nor, A. H. Abdullah, and M. B. Bashir, "A Discrete Firefly Algorithm for Scheduling Jobs on Computational Grid," in *Cuckoo Search and Firefly Algorithm*, ed: Springer, 2014, pp. 271-290.
- [10]. Kaur and S. Sharma, "Workflow scheduling in cloud computing environment using firefly algorithm," *International Journal of Computer Science and Technology*, vol. 8, pp. 73-76, 2017.
- [11]. I. Strumberger, M. Beko, M. Tuba, M. Minovic, and N. Bacanin, "Elephant herding optimization algorithm for wireless sensor network localization problem," in *Technological Innovation for Resilient Systems*, (Cham), pp. 175-184, Springer International Publishing, 2018.
- [12]. E. Tuba, A. Alihodzic, and M. Tuba, "Multilevel image thresholding using elephant herding optimization algorithm," in *Proceedings of 14th International Conference on the Engineering of Modern Electric Systems (EMES)*, pp. 240-243, June 2017.
- [13]. Muthusamy H, Ravindran S, Yaacob S, Polat K (2021) An improved elephant herding optimization using sine-cosine mechanism and opposition based learning for global optimization problems. *Expert Syst Appl* 172:114607
- [14]. Xavier, V.M.A.; Annadurai, S. Chaotic social spider algorithm for load balance aware task scheduling in cloud computing. *Clust. Comput.* 2019, 22, 287-297.
- [15]. Baş E, Ulker E (2020) A binary social spider algorithm for continuous optimization task. *Soft Comput* 24(17):12953-12979
- [16]. Zhou Y, Zhou Y, Luo Q, Abdel-Basset M (2017) A simplex method-based social spider optimization algorithm for clustering analysis. *Eng Appl Artif Intell* 64:67-82
- [17]. Mahato DP, Singh RS (2018) On maximizing reliability of grid transaction processing system considering balanced task allocation using social spider optimization. *Swarm Evol Comput* 38:202-217
- [18]. B. Kruekaew and W. Kimpan, Enhancing of Artificial Bee Colony Algorithm for Virtual Machine Scheduling and Load Balancing Problem in Cloud Computing, *International Journal of Computational Intelligence Systems.*, 13, 496-510 (2020).
- [19]. M. H. Kashani, M. Jamei, M. Akbari and R. M. Tayebi, *Utilizing Bee Colony to Solve Task Scheduling Problem in Distributed Systems*, 2011 Third International Conference on Computational Intelligence, Communication Systems and Networks., 298-303 (2011).
- [20]. N. Hesabian, H. Haj, and S. Javadi, "Optimal scheduling in cloud computing environment using the Bee algorithm," *International Journal of Computer Networks and Communications Security*, vol. 3, no. 6, pp. 253-258, June 2015.
- [21]. K. Thenmozhi, s. Udhaya, n. Vinothini, v. Nagaraju , "An Adaptive WSN clustering scheme using lion optimization algorithm to maintain coverage area in wireless sensor network", *international journal on recent researches in science, engineering & technology (ijrrset)*, 2019, 6(12), 6-12
- [22]. Natesan, G.; Chokkalingam, A. An improved grey wolf optimization algorithm based task scheduling in cloud computing environment. *Int. Arab J. Inf. Technol.* 2020, 17, 73-81.
- [23]. K.Vinothkumar, Dr.D.Maruthanayagam, "Investigation Of Optimal Task Offloading and Resource Allocation using Hybrid Grey Wolf Lion Optimization (HGWLO) in Cloud-Edge Computing", *Journal of Theoretical and Applied Information Technology*, ISSN: 1992-8645 E-ISSN: 1817-3195, Vol.101. No 14, July-2023, pp-5629-5644.
- [24]. Sobhanayak, S.; Kumar, T.A.; Bibhudatta, S. Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm. *Future Comput. Inform. J.* 2018, 3, 210-230.
- [25]. J. Verma, S. Sobhanayak, S. Sharma, A. Kumar-Turuk, and B. Sahoo, "Bacteria foraging based task scheduling algorithm in cloud computing environment,"

- International Conference on Computing, Communication and Automation (ICCCA), pp. 777–782, 2017.
- [26]. L. Jacob, V. Jeykrishanan, and P. Sengottuvelan, “Resource scheduling in cloud using bacterial foraging optimization algorithm,” *International Journal of Computer Applications*, vol. 92, no. 1, pp. 14–20, April 2014.