

COMPARATIVE PERFORMANCE EVALUATION OF BEE COLONY, BACTERIAL FORAGING, AND GREY WOLF OPTIMIZATION FOR TASK OFFLOADING AND RESOURCE SCHEDULING IN EDGE CLOUD ENVIRONMENTS

K.Vinothkumar,

Research Scholar,

PG and Research Department of Computer Science,

Sri Vijay Vidyalaya College of Arts & Science,

Dharmapuri, Tamilnadu, India

Dr. D. Maruthanayagam,

Professor,

Department of Computer Science,

AVS College of Arts & Science (Autonomous),

Salem, Tamilnadu, India

Abstract: Task offloading and resource scheduling are fundamental challenges in EdgeCloud computing, where computational tasks generated by resource-constrained mobile and edge devices must be efficiently allocated to heterogeneous computing resources. Inefficient task allocation can result in increased makespan, communication cost, execution time, and underutilization of available computing resources. This research study presents a comparative evaluation of three standalone metaheuristic optimization algorithms, namely Bee Colony Optimization (BCO), Bacterial Foraging Optimization (BFO), and Grey Wolf Optimization (GWO), for task offloading and resource scheduling in EdgeCloud environments. Unlike hybrid optimization approaches, each algorithm is independently implemented and evaluated under identical experimental conditions to ensure a fair comparison. The algorithms are integrated into the CloudEdgeSim simulation framework, where EdgeCloud resources, mobile devices, base stations, cloudlets, hosts, virtual machines, and network characteristics are modeled. The experimental configuration considers 5 data centers, 10 hosts, 50 cloudlet virtual machines, and 100 offloading tasks, with defined computational, memory, bandwidth, and network parameters. The performance of BCO, BFO, and GWO is evaluated using four key criteria: resource utilization, makespan, communication cost, and execution time. Multiple workload scenarios and repeated simulation runs are considered to assess algorithmic stability, scalability, and convergence behavior. The comparative analysis identifies the strengths and limitations of each standalone optimization technique and determines their relative suitability for task offloading and resource scheduling in EdgeCloud environments. The findings provide a controlled experimental basis for selecting an appropriate standalone metaheuristic algorithm for efficient EdgeCloud resource management.

Keywords: EdgeCloud Computing, Task Offloading, Resource Scheduling, Bee Colony Optimization, Bacterial Foraging Optimization, Grey Wolf Optimization, CloudEdgeSim, Resource Utilization, Makespan, Communication Cost, Execution Time and Metaheuristic Optimization.

I. INTRODUCTION

The fast growth of Internet of Things (IoT) devices, mobile applications, autonomous systems, smart cities, and latency-sensitive services has generated an increasing demand for computational resources with low response time and high availability. Traditional cloud computing provides substantial processing and storage capabilities; however, transferring computation-intensive tasks from end devices to remote cloud data centers can introduce considerable network latency, bandwidth consumption, and communication overhead. Edge computing addresses these limitations by placing computational resources closer to data-generating devices. The integration of edge and cloud resources, commonly referred to as EdgeCloud computing, provides a distributed computing environment in which tasks can be executed locally, at nearby edge servers, or at remote cloud data centers according to application requirements and resource availability. Efficient management of these heterogeneous resources is therefore essential for achieving improved performance and quality of service in EdgeCloud environments [1].

Task offloading is one of the fundamental mechanisms for managing computational workloads in EdgeCloud systems. In a typical environment, mobile and IoT devices have limited processing power, memory, and energy resources, making local execution inefficient for computationally intensive applications. Task offloading enables such devices to transfer computational workloads to edge servers or cloud resources

with greater processing capabilities. However, selecting an appropriate execution resource for each task is a complex optimization problem because multiple factors must be considered simultaneously, including task processing requirements, resource capacity, memory availability, bandwidth, network latency, communication overhead, and workload distribution. An inappropriate offloading decision may result in resource imbalance, increased communication cost, longer execution time, and higher overall makespan. Consequently, effective task-resource mapping is required to maximize resource utilization while minimizing the time and communication overhead associated with task execution [2]. The resource scheduling problem becomes more challenging because EdgeCloud infrastructures are inherently heterogeneous and dynamic. Edge servers generally provide lower communication latency but have comparatively limited computational resources, whereas cloud data centers offer considerably larger computational capacity but may involve higher communication latency. Furthermore, the arrival and computational requirements of tasks can vary significantly over time [3]. Therefore, conventional scheduling strategies based on simple heuristics may not consistently identify high-quality task-resource assignments within a large search space. The task offloading problem can be formulated as an optimization problem in which a set of computational tasks must be mapped to available edge and cloud resources while satisfying resource constraints and optimizing one or more performance objectives. Metaheuristic optimization

algorithms are particularly attractive for such problems because they can efficiently explore large and complex solution spaces without requiring an exhaustive search.

Among the various metaheuristic techniques, Bee Colony Optimization (BCO) provides a population-based search mechanism inspired by the foraging behavior of honey bees. In the context of EdgeCloud scheduling, individual bees can represent candidate task-resource allocation solutions, while the search process progressively improves these solutions according to their fitness. The exploration and exploitation characteristics of BCO make it suitable for searching large scheduling spaces where numerous possible task-resource combinations exist. Its ability to distribute the search among multiple candidate solutions can potentially improve resource allocation and reduce scheduling inefficiencies [4]. Bacterial Foraging Optimization (BFO) is another population-based optimization technique inspired by the foraging behavior of bacteria. BFO uses mechanisms such as chemotaxis, swarming, reproduction, and elimination-dispersal to explore and exploit the optimization space. For EdgeCloud task scheduling, each bacterium can represent a candidate task-offloading solution, and the fitness of each solution can be determined according to scheduling performance. The movement and reproduction mechanisms enable BFO to search for improved task-resource mappings while maintaining population diversity. These characteristics make BFO a potential approach for addressing the complex resource scheduling problem in distributed computing environments [5]. Grey Wolf Optimization (GWO) is a population-based metaheuristic inspired by the social hierarchy and hunting behavior of grey wolves. GWO organizes candidate solutions according to a hierarchical structure consisting of alpha, beta, delta, and omega wolves. The leading solutions guide the search process, allowing the algorithm to progressively converge toward promising regions of the solution space. In an EdgeCloud scheduling scenario, a wolf can represent a possible task-resource allocation, while the quality of each wolf is determined by the corresponding scheduling fitness. The balance between exploration and exploitation in GWO provides a mechanism for searching complex task-offloading solutions while gradually focusing on high-quality resource assignments [6].

Although BCO, BFO, and GWO have characteristics that make them applicable to optimization problems, their relative effectiveness for EdgeCloud task offloading cannot be determined solely from their theoretical search mechanisms. Performance can vary according to workload characteristics, resource heterogeneity, network conditions, population configuration, and optimization objectives. Furthermore, comparisons reported under different experimental environments may not provide a reliable basis for determining which algorithm is more appropriate for a particular EdgeCloud scheduling problem. A controlled comparison in which the same tasks, resources, network configuration, simulation parameters, and termination conditions are provided to all algorithms can provide a more meaningful assessment of their relative performance.

Therefore, this analysis focuses on a standalone comparative evaluation of BCO, BFO, and GWO for task offloading and resource scheduling in EdgeCloud environments. No hybridization or combination of the three algorithms is introduced. Each optimization algorithm independently

generates task-resource allocation decisions and is evaluated under the same experimental conditions. The algorithms are implemented within the CloudEdgeSim simulation environment, which enables the modeling of mobile devices, base stations, cloudlets, hosts, virtual machines, edge resources, cloud resources, and network characteristics. The experimental environment considers computational tasks and heterogeneous resources while incorporating relevant processing, memory, bandwidth, and communication parameters.

The performance of the three standalone algorithms is evaluated using four primary criteria: resource utilization, makespan, communication cost, and execution time. Resource utilization measures how effectively the available computing capacity is utilized, while makespan represents the time required to complete the overall workload. Communication cost evaluates the overhead associated with transferring tasks and data between devices, edge resources, and cloud resources, whereas execution time measures the time required for task processing. These metrics provide complementary perspectives on scheduling performance and enable a comprehensive comparison of the three optimization techniques.

The main objective of this research is not to introduce a new hybrid optimization algorithm but to establish a systematic and reproducible experimental comparison of three established standalone optimization approaches. The study investigates how BCO, BFO, and GWO behave under identical EdgeCloud workloads and resource configurations and identifies their respective strengths and limitations. The results are intended to provide practical guidance for selecting an appropriate standalone metaheuristic technique for EdgeCloud task offloading and resource scheduling. In addition, the comparative analysis can provide a foundation for future investigations involving larger workloads, dynamic task arrivals, energy-aware scheduling, multi-objective optimization, and real-world EdgeCloud deployments.

II. RELATED WORK

2.1. EdgeCloud Computing and Resource Scheduling

The rapid growth of Internet of Things (IoT), 5G communication, mobile applications, intelligent transportation, healthcare, and other latency-sensitive applications has increased the demand for computing resources closer to end users. Although conventional cloud computing provides substantial processing and storage capabilities, centralized cloud infrastructures may introduce communication latency and bandwidth consumption when computation-intensive tasks are transferred from geographically distributed devices. EdgeCloud computing addresses this limitation by combining the computational capabilities of edge nodes and cloud data centers, enabling tasks to be processed at different levels of the computing infrastructure according to resource availability and application requirements. Asghar and Jung (2022) presented a comprehensive survey of scheduling techniques in edge-cloud environments and classified existing scheduling approaches into heuristic and metaheuristic categories. Their study highlighted that scheduling plays an important role in determining how delay-sensitive workloads are distributed between edge and cloud resources. The authors also identified quality of service, latency, performance, and fault tolerance as

important considerations in EdgeCloud scheduling. Similarly, a comprehensive review of edge resource scheduling reported that the increasing deployment of IoT and 5G applications has made efficient task scheduling essential for resource-constrained edge environments. The study categorized resource-scheduling approaches according to architectures, application requirements, algorithm paradigms, and optimization objectives, while also identifying load balancing and fairness as important considerations for future EdgeCloud resource management [7]. Ullah et al. (2023) investigated task offloading and resource allocation in edge-cloud networks and emphasized that offloading decisions depend on factors such as available computational resources, network conditions, device location, and task requirements. Their DDQN-based approach considered resource utilization and task constraints and demonstrated that intelligent resource allocation can improve task-offloading performance compared with heuristic approaches. These studies demonstrate that resource scheduling is a fundamental component of EdgeCloud computing. However, the diversity of scheduling strategies and optimization objectives makes direct comparison of individual metaheuristic algorithms important [8].

2.2 Task Offloading in EdgeCloud Environments

Task offloading enables computationally constrained mobile and IoT devices to transfer computational workloads to nearby edge servers or remote cloud resources. The primary objective is to determine the most appropriate execution location while satisfying computational, communication, and application constraints. In an EdgeCloud environment, an offloading decision can involve local execution, edge execution, or cloud execution. The decision directly influences execution time, communication overhead, resource utilization, and overall system performance.

Babar et al. (2021) proposed an Artificial Bee Colony (ABC)-based computation-offloading approach for scalable edge computing environments. Their framework integrated edge and cloud resources and addressed workload management under latency and resource constraints. The study reported improvements in response time and workload management compared with PSO, ACO, and Round-Robin scheduling approaches. The authors also emphasized that large-scale computation offloading creates resource congestion and therefore requires effective resource scheduling mechanisms [9]. Abbas et al. (2021) investigated metaheuristic-based task offloading in mobile edge computing and evaluated ACO, Whale Optimization, and Grey Wolf Optimization approaches. Their study considered the trade-off between energy consumption and execution delay and reported that GWO provided favorable performance for the investigated offloading problem. This work demonstrates the potential of population-based metaheuristics for selecting appropriate offloading tasks in resource-constrained edge environments [10]. Pang et al. (2021) investigated computing-resource scheduling for massive IoT devices in mobile edge computing. Their work considered task urgency and base-station energy and developed an optimization-based resource-scheduling strategy for massive IoT workloads. The results demonstrated the importance of intelligent resource allocation when a large number of IoT devices compete for limited edge resources [11]. More recently, Ullah et al. (2023) formulated edge-cloud task offloading as a multi-objective decision problem

involving computation and communication requirements. Their study modeled tasks using computational requirements, input/output data sizes, and latency constraints and considered both edge and cloud resources in the offloading decision.

2.3 Bee Colony Optimization for Edge Task Offloading

Bee-inspired optimization techniques have been increasingly investigated for resource management and task offloading because of their population-based exploration and exploitation characteristics. Babar et al. (2021) developed an Artificial Bee Colony-based computation-offloading framework for IoT applications in scalable edge computing. The authors modeled a three-tier edge-cloud architecture and used ABC to manage computational workloads while considering response time and resource utilization. Their results showed that ABC could achieve improved response time and workload balancing compared with PSO, ACO, and Round-Robin scheduling. A more recent study on vehicular edge computing proposed a bee-colony-based task-offloading algorithm for selecting suitable servers in dynamic vehicular environments. The study considered contextual and wireless-network parameters and focused on reducing application execution time and task failures. Its results indicated that bee-colony-based optimization can effectively address task scheduling in highly dynamic edge environments. These findings suggest that bee-inspired optimization can be useful for task-resource mapping in distributed edge environments. Nevertheless, the performance of bee-based optimization depends on workload characteristics, resource heterogeneity, and algorithm configuration. Therefore, evaluating it under the same experimental conditions as other standalone algorithms is necessary to establish a fair comparison [12].

2.4 Bacterial Foraging Optimization for Edge Task Offloading

Bacterial Foraging Optimization (BFO) is inspired by the foraging behavior of bacteria and uses mechanisms such as chemotaxis, reproduction, swarming, and elimination-dispersal to explore an optimization space. These characteristics make BFO potentially suitable for scheduling problems involving large and complex task-resource search spaces. Babar et al. (2022) investigated bacterial-foraging-based computation offloading for IoT sensors in edge computing. Their study proposed a multi-objective BFO-based offloading mechanism for load management across edge servers. The evaluation considered response time, communication cost, degree imbalance, and workload management and compared the approach with ACO, PSO, and Round-Robin scheduling. The results indicated that the BFO-based approach could reduce response time and communication cost while improving workload management [13].

2.5 Grey Wolf Optimization for Edge Task Offloading

Grey Wolf Optimization (GWO) has attracted considerable attention in task offloading because of its relatively simple population structure and exploration-exploitation mechanism. The algorithm models candidate solutions using a hierarchy of alpha, beta, delta, and omega wolves, with leading solutions guiding the search toward promising regions.

Abbas et al. (2021) evaluated GWO for metaheuristic-based task offloading in mobile edge computing alongside ACO and Whale Optimization. Their study considered the relationship

between energy consumption and execution delay and reported that GWO achieved favorable performance in selecting an optimized set of offloading tasks. **Li et al. (2022)** investigated an improved GWO-based task scheduling approach for edge computing in discrete manufacturing environments. The study modified the convergence mechanism and dynamic weighting strategy of GWO to improve optimization speed and accuracy. The experimental results demonstrated reductions in processing delay and resource consumption under the investigated workload conditions [14]. **Nujhat et al. (2024)** proposed a collaborative edge-computing task-offloading framework based on Binary Multi-Objective Grey Wolf Optimization. Their framework jointly considered latency, energy consumption, and resource usage cost in a collaborative edge environment. The authors formulated the offloading problem as a multi-objective optimization problem and demonstrated improvements in latency, energy, and resource cost compared with selected existing approaches [15].

The reviewed literature demonstrates that metaheuristic and intelligent optimization approaches can improve task offloading and resource scheduling in edge and EdgeCloud environments. Existing research has investigated a wide range of objectives, including latency, energy consumption, resource utilization, communication cost, task completion time, and workload balancing. For example, **Babar et al. (2021)** focused on ABC-based resource management and response-time improvement, whereas **Babar et al. (2022)** investigated BFO-based smart offloading with emphasis on response time, communication cost, and load management. **Abbas et al. (2021)** examined GWO together with other metaheuristics for energy and delay optimization, while **Nujhat et al. (2024)** investigated a binary multi-objective GWO formulation for latency, energy, and resource cost. Consequently, the results of these studies cannot be interpreted as a direct ranking of BCO, BFO, and GWO because they were conducted under different system assumptions and objectives. A controlled experimental comparison is therefore necessary.

III. SYSTEM MODEL

This research analysis considers an EdgeCloud computing environment in which computational tasks generated by mobile devices are processed either locally or offloaded to nearby edge resources or centralized cloud resources. The system consists of mobile devices (MDs), base stations (BSs), edge servers/cloudlets, virtual machines (VMs), and cloud data centers. The overall architecture enables computational workloads to move from resource-constrained end devices toward more powerful edge and cloud resources. The task-offloading decision is performed independently using Bee Colony Optimization (BCO), Bacterial Foraging Optimization (BFO), and Grey Wolf Optimization (GWO). No hybridization is performed among these algorithms [16].

The considered EdgeCloud architecture follows a hierarchical structure consisting of four principal layers: mobile devices, base stations, edge computing resources, and cloud data centers, as shown in Figure 1. Mobile devices generate computational tasks and communicate with the infrastructure through base stations. The base stations provide connectivity between mobile devices and nearby edge computing resources. Edge servers or cloudlets provide relatively low-latency computation because they are positioned closer to the task-generating devices. Computationally intensive workloads

that cannot be efficiently processed at the edge can subsequently be assigned to cloud data centers. The proposed system does not force every task to follow all layers sequentially. Instead, a task can be processed locally or offloaded to an appropriate edge or cloud resource depending on the resource availability and scheduling decision. The optimization algorithms determine the task-resource assignment while considering computational and communication characteristics [17].

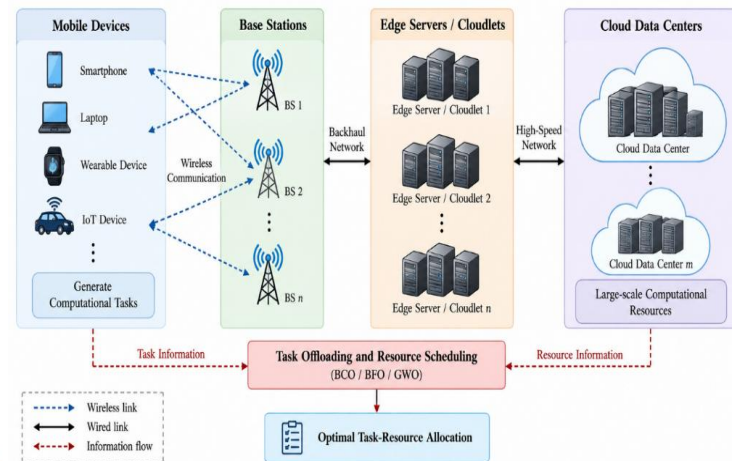


Figure 1: EdgeCloud architecture for task offloading and resource scheduling

3.1. System Components

- **Mobile Devices:** Mobile devices (MDs) represent IoT-enabled or mobile computing devices that generate computational tasks. Each device has limited computational resources compared with edge servers and cloud data centers. **A task generated by a mobile device may either be processed locally or transferred to an edge/cloud resource for remote execution.** Each task T_i is associated with computational and communication requirements, including task length, input data size, output data size, and processing requirements.
- **Base Stations:** Base stations (BSs) provide wireless communication between mobile devices and the EdgeCloud infrastructure. They act as communication access points through which computational tasks and associated data are transmitted to edge resources. Network characteristics such as bandwidth and communication latency influence the cost and time required for task offloading.
- **Edge Servers/Cloudlets:** Edge servers or cloudlets provide computing resources close to mobile devices. Their proximity enables lower communication latency compared with remote cloud data centers. Edge resources are therefore suitable for tasks requiring rapid processing and low response time. However, their computational capacity may be more limited than that of centralized cloud resources.
- **Cloud Data Centers:** Cloud data centers provide large-scale computational and storage resources. They can execute computationally intensive tasks when sufficient edge resources are unavailable. However, communication between mobile devices/edge resources and remote cloud data centers may introduce additional network overhead [18].

- **Virtual Machines: Virtual machines (VMs) provide virtualized computing environments for executing offloaded tasks.** Each VM is characterized by computational capacity, memory, and bandwidth. The task scheduler maps computational tasks to suitable VMs according to the optimization solution generated by BCO, BFO, or GWO.

3.2. Problem Formulation

Task offloading and resource scheduling are formulated as a constrained task-resource assignment problem. Let the set of computational tasks be represented by:

$$T = \{T_1, T_2, \dots, T_n\}$$

where n denotes the total number of computational tasks.

Let the available computing resources be represented by:

$$R = \{R_1, R_2, \dots, R_m\}$$

where m denotes the total number of available computing resources, including edge and cloud resources.

The task-resource assignment variable is defined as:

$$X_{ij} = \begin{cases} 1, & \text{if task } T_i \text{ is assigned to resource } R_j \\ 0, & \text{Otherwise} \end{cases}$$

The assignment constraint is expressed as:

$$\sum_{j=1}^m X_{ij} = 1, \quad \forall i \in \{1, \dots, n\}$$

which ensures that each task is assigned to exactly one execution resource.

The resource capacity constraint can be expressed as:

$$\sum_{i=1}^n X_{ij} C_i \leq C_j^{\max}, \quad \forall j \in \{1, \dots, m\}$$

where C_i represents the computational requirement of task T_i , and $C_j^{\max}$ represents the available computational capacity of resource R_j . Similarly, the memory requirement must satisfy:

$$\sum_{i=1}^n X_{ij} Mem_i \leq Mem_j^{\max}$$

where Mem_i is the memory requirement of task T_i , and $Mem_j^{\max}$ is the available memory of resource R_j . These constraints ensure that the generated schedules are feasible with respect to the available computational and memory resources [19].

3.3. Task Model

Let each computational task T_i be characterized by:

$$T_i = \{L_i, CPU_i, Mem_i, D_i^{\text{in}}, D_i^{\text{out}}\}$$

Where, L_i = computational length of task T_i ; CPU_i = CPU requirement; Mem_i = memory requirement; D_i^{in} = input data size; D_i^{out} = output data size. The computational execution time of task T_i on resource R_j can be represented as:

$$ET_{ij} = \frac{L_i}{MIPS_j}$$

where $MIPS_j$ denotes the processing capacity of resource R_j . The total task completion time may additionally include communication time:

$$CT_{ij} = TT_{ij} + ET_{ij}$$

where TT_{ij} denotes the transmission time between the task-generating device and the selected computing resource. For a network connection with bandwidth BW_{ij} , transmission time can be represented as:

$$TT_{ij} = \frac{D_i^{\text{in}} + D_i^{\text{out}}}{BW_{ij}}$$

when both input and output data transfer are included in the communication model.

3.4. Resource Model

Each computing resource R_j is represented as:

$$R_j = \{MIPS_j, Mem_j, Storage_j, BW_j\}$$

Where, $MIPS_j$ = processing capacity; Mem_j = available memory; $Storage_j$ = storage capacity; BW_j = communication bandwidth. The EdgeCloud infrastructure consists of heterogeneous resources. Therefore, different resources may have different processing capacities, memory sizes, and bandwidth capabilities. This heterogeneity increases the complexity of the task scheduling problem because assigning a task to a resource with insufficient capacity can lead to resource contention or infeasible execution [20].

3.5. Task Offloading Decision

For each task T_i , the execution location is determined by an offloading decision:

$$O_i \in \{Local, Edge, Cloud\}$$

Where, Local indicates execution on the originating mobile device; Edge indicates execution on an edge server/cloudlet; Cloud indicates execution at a cloud data center. For tasks selected for offloading, the optimization algorithm determines the specific resource R_j to which the task is assigned. Thus, the overall scheduling solution can be represented as:

$$S = [R_{j_1}, R_{j_2}, \dots, R_{j_n}]$$

Where, R_{j_i} represents the selected execution resource for task T_i . BCO, BFO, and GWO independently search the solution space to identify high-quality task-resource mappings. The effectiveness of the task-offloading algorithms is evaluated using four principal performance measures: Resource utilization, Makespan, Communication cost and Execution time. These metrics are selected because they collectively measure resource efficiency, workload completion, communication overhead, and computational performance [21].

IV. METHODOLOGY

This work independently evaluates Bee Colony Optimization (BCO), Bacterial Foraging Optimization (BFO), and Grey Wolf Optimization (GWO) for task offloading and resource scheduling in EdgeCloud environments. Each algorithm operates as a standalone optimization approach, using the same tasks, resources, network configuration, and simulation conditions. No hybridization or interaction among BCO, BFO, and GWO is employed, enabling a fair and consistent performance comparison.

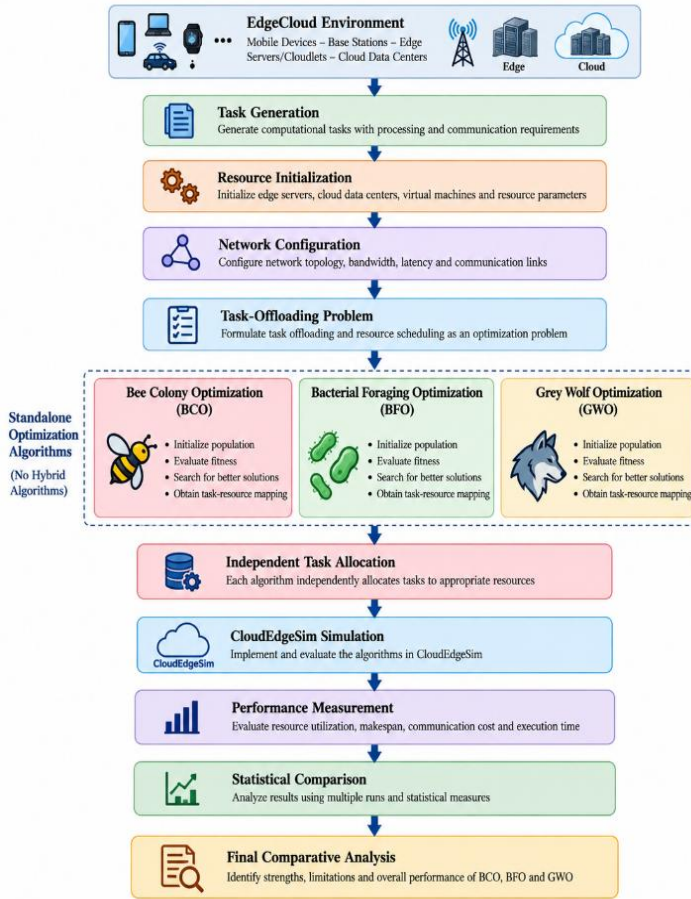


Figure 2: Flow of EdgeCloud Task Offloading Methodology

The methodology begins by defining the EdgeCloud environment, including mobile devices, edge servers, cloudlets, and cloud data centers. Computational tasks are then generated, followed by initialization of the available computing resources and network configuration. The task-offloading problem is formulated as an optimization problem and independently solved using BCO, BFO, and GWO. Each algorithm performs independent task-resource allocation without any hybridization. The resulting schedules are evaluated using CloudEdgeSim, and their performance is measured in terms of resource utilization, makespan, communication cost, and execution time [22]. Finally, statistical analysis is performed to compare the three standalone algorithms and identify their relative strengths and overall performance.

4.1. Bee Colony Optimization for Task Offloading

Bee Colony Optimization (BCO) is a population-based metaheuristic optimization technique inspired by the cooperative foraging behavior of honey bees. In nature, bees search for food sources, evaluate their quality, and communicate information about promising sources to other members of the colony. **The search process therefore combines exploration, where new food sources are discovered, and exploitation, where promising food sources are investigated further.** In the proposed EdgeCloud task-offloading problem, each bee represents a candidate task-resource scheduling solution. A food source corresponds to a possible task-resource allocation, and the quality of the food source represents the fitness of the corresponding scheduling

solution [23]. Through iterative evaluation and modification of candidate solutions, BCO searches for an efficient allocation of computational tasks to available EdgeCloud resources. BCO Algorithms Steps are followed,

Step 1: BCO Initialization

Step 1.1: Define the population: The first step is to initialize a population of P bees. Each bee represents one feasible task-resource allocation. Let the initial population be:

$$B = \{S_1, S_2, \dots, S_P\}$$

where P is the number of bees and S_k is the candidate solution represented by bee k.

Step 1.2: Generate task-resource assignments

For each bee, a resource is selected for every task:

$$S_k = [R_{j1}, R_{j2}, \dots, R_{jn}]$$

where:

$$R_{ji} \in R$$

The resource assignment must satisfy the available resource constraints. For example, if there are five resources and four tasks, one possible initial solution is:

$$S_1 = [R_2, R_5, R_1, R_4]$$

Another bee may represent:

$$S_2 = [R_1, R_3, R_5, R_2]$$

Thus, different bees explore different task-resource mappings. The initialization process provides the starting population for BCO. Maintaining diverse feasible task-resource mappings is important because it allows the algorithm to explore different regions of the scheduling search space. A larger and more diverse initial population increases the possibility of discovering high-quality task allocations.

Step 2: Fitness Evaluation

After initialization, each bee solution is evaluated according to the defined EdgeCloud scheduling objectives. The fitness function measures the quality of the task-resource allocation. The four evaluation criteria are: Resource utilization; Makespan; Communication cost; and Execution time. Because resource utilization is maximized while the other three criteria are minimized, the metrics should be normalized before constructing a combined fitness function. For a metric x that must be minimized:

$$x^{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

For resource utilization, which is maximized:

$$RU^{norm} = \frac{RU - RU_{min}}{RU_{max} - RU_{min}}$$

A minimization-oriented fitness function can then be defined as:

$$F(S) = \omega_1 M^{norm} + \omega_2 CC^{norm} + \omega_3 ET^{norm} + \omega_4 (1 - RU^{norm})$$

Where, M^{norm} = normalized makespan; CC^{norm} = normalized communication cost; ET^{norm} = normalized execution time; RU^{norm} = normalized resource utilization; $\omega_1, \omega_2, \omega_3, \omega_4$ = objective weights.

The weights satisfy: $\omega_1 + \omega_2 + \omega_3 + \omega_4 = 1$

If equal importance is assigned to all four objectives: $\omega_1 = \omega_2 = \omega_3 = \omega_4 = 0.25$

Fitness-based selection

The best solution is obtained by:

$$S_{best} = \arg \min_{S_k} F(S_k)$$

A smaller fitness value indicates a better overall scheduling solution.

Fitness evaluation converts the performance of each task-resource allocation into a single value that can be used by BCO to compare candidate solutions. A solution with lower makespan, communication cost, and execution time and higher resource utilization receives a better fitness value. The fitness function therefore provides the mechanism through which BCO determines which candidate schedules should be retained and further explored. Use the weighted equation above only if your actual implementation uses a combined fitness function. If your implementation evaluates the four metrics separately, the fitness formulation should be changed accordingly.

Step 3: Exploration and Exploitation

BCO improves the candidate task-resource allocations through two fundamental search behaviors: **exploration** and **exploitation**.

Step 3.1: Exploration

Exploration allows bees to investigate new task-resource combinations rather than repeatedly selecting the same resources. A neighboring solution can be generated by changing the resource assignment of one or more tasks:

$$S'_k = [R'_{j1}, R'_{j2}, \dots, R'_{jn}]$$

where one or more R'_{ji} differ from the original solution.

Step 3.2: Exploitation

Exploitation focuses the search around promising solutions. **If a bee discovers a high-quality task-resource allocation, neighboring solutions are generated around that allocation to find further improvements.** Let the probability of selecting a promising solution be represented by:

$$P_k = \frac{Q_k}{\sum_{l=1}^P Q_l}$$

where Q_k represents the quality of bee k . A common quality transformation for a minimization problem is:

$$Q_k = \frac{1}{F(S_k) + \epsilon}$$

where ϵ is a small positive value used to avoid division by zero. Thus, solutions with smaller fitness values obtain higher selection probabilities.

Step 4: Best-solution update

After generating and evaluating new solutions:

$$S_{best}^{new} = \begin{cases} S'_k, & F(S'_k) < F(S_{best}) \\ S_{best}, & \text{otherwise} \end{cases}$$

The exploration mechanism prevents the algorithm from searching only around previously identified solutions, while exploitation concentrates computational effort on promising task-resource mappings. The balance between these two behaviors enables BCO to search the large EdgeCloud scheduling space efficiently. Exploration increases solution diversity, whereas exploitation improves already promising allocations. The process continues iteratively until the predefined maximum number of iterations is reached or the stopping condition is satisfied.

Algorithm 1: Standalone BCO-Based Task Offloading and Resource Scheduling

Input:

T = Set of computational tasks

R = Set of EdgeCloud resources

P = Number of bees

I_{max} = Maximum number of iterations

Output:

S_{best} = Best task-resource allocation

1. Initialize P feasible bee solutions.
2. Represent each bee as a task-resource mapping.
3. Evaluate the fitness of every bee.
4. Identify the initial best solution S_{best} .
5. Set iteration counter $k = 1$.
6. While $k \leq I_{max}$ do
7. Select promising bee solutions according to their fitness.
8. Generate neighboring task-resource allocations for selected bees.
9. Check computational and memory constraints.
10. Evaluate the fitness of the newly generated solutions.
11. If the new solution has better fitness:
 Replace the corresponding previous solution.
12. Perform exploration by generating alternative task-resource mappings.
13. Evaluate the newly explored solutions.
14. Update S_{best} if a better feasible solution is found.
15. $k = k + 1$.
16. End While
17. Return S_{best} .
18. Execute S_{best} in CloudEdgeSim.
19. Record:
 Resource Utilization,
 Makespan,
 Communication Cost,
 Execution Time.

The BCO-based task-offloading algorithm starts by generating a population of feasible bees, where each bee represents an independent task-resource allocation across the available EdgeCloud resources. Each candidate solution is evaluated according to the defined scheduling fitness function, which incorporates resource utilization, makespan, communication cost, and execution time. The best solution obtained from the initial population is stored as the current global solution. During each optimization iteration, promising bee solutions are selected for exploitation, and neighboring task-resource mappings are generated by changing the resource assignments of selected tasks. The new solutions are checked against computational and memory constraints and then evaluated using the fitness function. Solutions that provide better scheduling performance are retained, while an exploration process generates additional task-resource combinations to maintain diversity in the population. The current best solution is updated whenever a superior feasible allocation is identified. This exploration and exploitation process is repeated until the maximum number of iterations is reached. Finally, the best task-resource mapping obtained by the standalone BCO algorithm is submitted to CloudEdgeSim for execution, and the resulting resource utilization, makespan, communication cost, and execution time are recorded. The same experimental procedure is subsequently applied independently to BFO and

GWO so that the three algorithms can be compared under identical EdgeCloud conditions.

4.2. Bacterial Foraging Optimization for Task Offloading

Bacterial Foraging Optimization (BFO) is used in this study as a standalone optimization algorithm for task offloading and resource scheduling. The BFO process is independently applied to the EdgeCloud scheduling problem without combining it with BCO, GWO, or any other optimization technique. Each bacterium represents a candidate task-resource allocation, and the bacterial population iteratively improves these allocations through chemotaxis, swarming, reproduction, and elimination-dispersal mechanisms. Bacterial Foraging Optimization is a population-based metaheuristic inspired by the foraging behavior of bacteria. In a natural environment, bacteria search for nutrient-rich regions by changing their movement direction, interacting with neighboring bacteria, reproducing when sufficient nutrients are obtained, and dispersing when necessary. These behaviors provide mechanisms for exploration and exploitation of a search space. In the proposed EdgeCloud task-offloading problem, each bacterium represents a possible allocation of computational tasks to available resources [24][25]. The quality of a bacterium is determined by the fitness of its corresponding scheduling solution. The BFO algorithm progressively improves the task-resource allocation through chemotaxis, swarming, reproduction, and elimination-dispersal operations. The objective is to identify a feasible allocation that provides efficient resource utilization while minimizing makespan, communication cost, and execution time.

BFO Algorithm steps are followed,

Step 1: Initialization

The BFO process starts by generating a population of P bacteria. Let the bacterial population be:

$$B = \{B_1, B_2, \dots, B_p\}$$

where PP is the population size.

Each bacterium is initialized with a randomly generated feasible task-resource allocation:

$$B_k = [R_{j1}, R_{j2}, \dots, R_{jn}]$$

for $k=1,2,\dots,P$.

The generated solutions are checked against computational and memory constraints. If an allocation violates a resource constraint, the corresponding task is reassigned to another feasible resource.

The fitness of each initial bacterium is then calculated:

$$F(B_k) = \omega_1 M_k^{norm} + \omega_2 CC_k^{norm} + \omega_3 ET_k^{norm} + \omega_4 (1 - RU_k^{norm})$$

where M_k , CC_k , ET_k , and RU_k represent makespan, communication cost, execution time, and resource utilization for bacterium k .

The best initial bacterium is stored as:

$$B_{best} = \arg \min_{B_k} F(B_k)$$

Initialization creates a diverse set of possible scheduling solutions. Different bacteria explore different task-resource combinations, providing the starting points for the subsequent chemotaxis and optimization operations.

Step 2: Chemotaxis

Chemotaxis represents the movement of bacteria toward more favorable nutrient regions. In the scheduling problem, it

corresponds to modifying the current task-resource assignment to search for an improved solution.

For a current bacterium B_k , a neighboring solution is generated as:

$$B'_k = B_k + \Delta_k$$

where Δ_k represents a discrete change in one or more task-resource assignments.

For example:

$$B_k = [R_2, R_5, R_1, R_4]$$

may be changed to:

$$B'_k = [R_2, R_5, R_1, R_4]$$

Here, the resource assigned to T2 is changed from R5 to R3.

The new solution is evaluated:

$$F(B'_k)$$

and compared with the current solution:

$$B_k = \begin{cases} B'_k & F(B'_k) < F(B_k) \\ B_k & \text{Otherwise} \end{cases}$$

Chemotaxis enables BFO to search the neighborhood of the current task-resource allocation. **By repeatedly changing resource assignments and evaluating the resulting schedules**, the algorithm can identify improved allocations with lower scheduling cost.

Step 3: Swarming

The swarming mechanism models the tendency of bacteria to communicate and form groups around favorable nutrient regions. In the task-offloading problem, promising scheduling solutions influence the search toward high-quality regions. A simplified swarming component can be represented as:

$$S(B_k) = \sum_{l=1}^P Q(B_k, B_l)$$

where $Q(B_k, B_l)$ represents the interaction between bacteria B_k and B_l . The fitness after incorporating the swarming behavior can be expressed as:

$$F_{swarm}(B_k) = F(B_k) + S(B_k)$$

The interaction encourages the population to search around promising candidate solutions while maintaining relationships among the bacterial population. **Swarming helps BFO exploit high-quality regions of the task-resource search space. Instead of optimizing each candidate solution completely independently**, information from neighboring bacteria influences the search direction. This can improve convergence toward promising scheduling allocations.

Step 4: Reproduction

After a predefined number of chemotactic movements, the bacteria are ranked according to their fitness values. Bacteria with better fitness values are considered healthier and are retained, while poorer solutions are eliminated.

If the population is sorted as:

$$F(B_1) \leq F(B_2) \leq \dots \leq F(B_p)$$

the better-performing bacteria are selected for reproduction.

The population is then regenerated by duplicating selected high-quality bacteria:

$$B_k^{new} = B_k^{selected}$$

Reproduction increases the representation of promising task-resource allocations in the population. Poor-performing solutions are removed, allowing computational effort to focus on better scheduling configurations. This mechanism represents the exploitation component of BFO.

Step 5: Elimination-Dispersal

Elimination-dispersal introduces diversity into the bacterial population and reduces the possibility of premature convergence.

For each bacterium, an elimination-dispersal probability P_{ed} is considered:

$$B_k = \begin{cases} B_k^{new} \text{rand} < P_{ed} \\ B_k \text{rand} \geq P_{ed} \end{cases}$$

where rand is a uniformly generated random number between 0 and 1. If elimination occurs, the affected bacterium is replaced by a newly generated feasible task-resource allocation. Elimination-dispersal enables BFO to escape from potentially poor local optima. By replacing selected bacteria with new feasible solutions, the algorithm introduces new task-resource combinations into the search process. This maintains population diversity and improves exploration of the scheduling search space.

BFO Task-Offloading Algorithm

Algorithm 2: Standalone BFO-Based Task Offloading and Resource Scheduling

Input:

- T = Set of computational tasks
- R = Set of EdgeCloud resources
- P = Bacterial population size
- N_c = Number of chemotactic steps
- N_s = Number of swim steps
- N_{re} = Number of reproduction steps
- N_{ed} = Number of elimination-dispersal events
- P_{ed} = Elimination-dispersal probability

Output:

B_{best} = Best task-resource allocation

1. Generate P feasible bacterial solutions.
2. Represent each bacterium as a task-resource mapping.
3. Evaluate the fitness of every bacterium.
4. Identify the initial best solution B_{best} .
5. For elimination-dispersal cycle = 1 to N_{ed} do
6. For reproduction cycle = 1 to N_{re} do
7. For each bacterium B_k do
8. Evaluate current fitness $F(B_k)$.
9. Perform chemotactic movement by modifying one or more task-resource assignments.
10. Check resource constraints.
11. Evaluate the new solution.
12. If the new solution is better:
Accept the new solution.
13. Perform swim/search steps around the improved solution.
14. Apply swarming interaction among bacteria.
15. Update B_{best} if a better feasible solution is obtained.
16. End For
17. Rank bacteria according to fitness.
18. Retain better bacteria for reproduction.
19. Duplicate selected high-quality bacteria to restore the population.
20. End For
21. Apply elimination-dispersal probability P_{ed} .
22. Replace selected bacteria with newly generated feasible task-resource allocations.
23. Re-evaluate the population.
24. Update B_{best} .

25. End For
26. Return B_{best} .
27. Execute B_{best} in CloudEdgeSim.
28. Record:
Resource Utilization,
Makespan,
Communication Cost,
Execution Time.

The BFO-based task-offloading algorithm begins by generating a population of feasible bacteria, where each bacterium represents an independent assignment of computational tasks to available EdgeCloud resources. The initial bacterial population is evaluated using the defined fitness function based on resource utilization, makespan, communication cost, and execution time, and the best-performing allocation is stored as the current best solution. During the chemotaxis stage, individual bacteria modify one or more task-resource assignments to generate neighboring solutions and move toward allocations with improved fitness. The swarming mechanism enables promising solutions to influence the search behavior of other bacteria, thereby encouraging the population to investigate favorable regions of the scheduling search space. After a predefined number of chemotactic steps, the bacteria are ranked according to their fitness, and better-performing bacteria are retained and reproduced, while weaker solutions are discarded. The elimination-dispersal mechanism is subsequently applied to introduce new feasible task-resource allocations and maintain population diversity. The resulting population is again evaluated, and the global best allocation is updated whenever a superior feasible solution is discovered. These chemotaxis, swarming, reproduction, and elimination-dispersal processes are repeated until the predefined termination condition is satisfied. Finally, the best task-resource allocation generated by the standalone BFO algorithm is executed in CloudEdgeSim, and its resource utilization, makespan, communication cost, and execution time are recorded. The same workload and EdgeCloud configuration are then used independently for BCO and GWO to ensure a fair comparative evaluation.

4.3. Grey Wolf Optimization for Task Offloading

Grey Wolf Optimization (GWO) is employed in this study as a population-based optimization algorithm for task offloading and resource scheduling in the EdgeCloud environment. GWO models the social hierarchy and hunting behavior of grey wolves to search for an optimal solution [26]. In the framework, each wolf represents a candidate task-resource allocation, and the optimization process independently searches for a high-quality schedule without using information from BCO or BFO.

GWO Principle: Grey Wolf Optimization is inspired by the social hierarchy and cooperative hunting mechanism of grey wolves. The wolf population is organized into four hierarchical levels: alpha (α), beta (β), delta (δ), and omega (ω).

- The alpha wolf represents the best candidate solution identified in the current population. In the EdgeCloud scheduling problem, the alpha solution corresponds to the task-resource allocation with the best fitness.

- The beta wolf represents the second-best candidate solution and assists the alpha in guiding the search toward promising regions.
- The delta wolf represents the third-best candidate solution and provides an additional guidance point during the optimization process.
- The remaining candidate solutions are considered omega wolves. They update their positions according to the guidance provided by the alpha, beta, and delta wolves.

Thus, the hierarchy can be represented as:

$$\alpha \rightarrow \beta \rightarrow \delta \rightarrow \omega$$

where α , β , and δ guide the optimization of the remaining wolves. **The hunting behavior of GWO provides a mechanism for balancing exploration and exploitation.** At the beginning of optimization, the algorithm explores different regions of the solution space, while later iterations increasingly concentrate the search around promising solutions.

GWO Algorithms Steps as followed,

Step 1: Initialization

The GWO process begins by generating an initial population of P wolves. Each wolf is assigned a feasible task-resource mapping.

The initial population is given by:

$$W = \{W_1, W_2, \dots, W_p\}$$

Each wolf is generated as:

$$W_k = [R_{j1}, R_{j2}, \dots, R_{jn}]$$

where each task is assigned to an available resource while satisfying the computational and memory constraints.

After generating the initial population, the fitness of every wolf is calculated.

The best three solutions are identified as:

$$\begin{aligned} W_\alpha &= \argmin F(W_k) \\ W_\beta &= \text{second - best solution} \\ W_\delta &= \text{third best solution.} \end{aligned}$$

All remaining wolves are classified as omega wolves. Initialization creates a diverse set of possible task-resource allocations. Since different wolves represent different schedules, the population can investigate multiple regions of the scheduling search space. The three best initial solutions become the leaders that guide the remaining wolves during subsequent iterations.

Step 2: Fitness Evaluation

Each wolf is evaluated according to the performance of its corresponding task-resource allocation. The four performance criteria considered in this study are: Resource utilization; Makespan; Communication cost; and Execution time. If a combined normalized objective is used, the fitness function can be represented as:

$$F(B_k) = \omega_1 M_k^{\text{norm}} + \omega_2 CC_k^{\text{norm}} + \omega_3 ET_k^{\text{norm}} + \omega_4 (1 - RU_k^{\text{norm}})$$

Where, M_k^{norm} = normalized makespan; CC_k^{norm} = normalized communication cost; ET_k^{norm} = normalized execution time; RU_k^{norm} = normalized resource utilization; w_1, w_2, w_3, w_4 = objective weights. The weights satisfy:

$$\sum_{q=1}^4 \omega_q = 1$$

The objective is to minimize: $\min F(W_k)$. The wolf with the smallest fitness value is considered the alpha solution. **Fitness evaluation allows GWO to determine the quality of each candidate schedule.** A solution with lower makespan, communication cost, and execution time and efficient resource utilization receives a better fitness value. The fitness values are also used to establish the alpha, beta, and delta hierarchy.

Step 3: Hierarchical Search

After fitness evaluation, the three best wolves are selected as:

$$W_\alpha, W_\beta, W_\delta$$

These three solutions guide the remaining wolves. The alpha wolf provides the strongest guidance because it represents the best solution currently available. The beta wolf provides the second level of guidance, while the delta wolf provides the third level. Omega wolves update their positions based on the three leaders. Conceptually:

$$W_\omega \leftarrow \{W_\alpha, W_\beta, W_\delta\}$$

This hierarchical mechanism prevents the search from depending on only a single candidate solution. Instead, multiple high-quality task-resource allocations guide the optimization process. The hierarchical search enables GWO to exploit promising regions of the task-resource search space while retaining information from several high-quality solutions. This is particularly useful in EdgeCloud scheduling because several different resource mappings may provide competitive scheduling performance.

Step 4: Position Update

The principal GWO position-update mechanism models the encircling and hunting behavior of wolves.

For each wolf, the distance from the alpha solution is calculated as:

$$D_\alpha = |C_1 \cdot W_\alpha W|$$

Similarly:

$$D_\beta = |C_2 \cdot W_\beta W|$$

And

$$D_\delta = |C_3 \cdot W_\delta W|$$

The corresponding candidate positions are calculated as:

$$W_1 = W_\alpha - A_1 \cdot D_\alpha$$

$$W_2 = W_\beta - A_2 \cdot D_\beta$$

$$W_3 = W_\delta - A_3 \cdot D_\delta$$

The updated wolf position is then calculated as:

$$W(t+1) = \frac{W_1 + W_2 + W_3}{3}$$

The coefficient vectors are defined as:

$$A = 2ar_1 - a$$

and

$$C = 2r_2$$

where r_1 and r_2 are random values in $[0,1]$, and aa decreases linearly during the optimization process:

where:

- t = current iteration;
- $T_{\max}[\bar{f}_0]$ = maximum number of iterations.

Step 5: Exploration and exploitation

When $|A| > 1$, the wolves are encouraged to explore different regions of the solution space. When $|A| < 1$, the wolves move closer to the leading solutions, increasing exploitation. Therefore, the decreasing aa parameter provides a transition from:

$$\text{Exploration} \rightarrow \text{Exploitation}$$

Step 6: Discrete task-resource mapping

Because EdgeCloud resource allocation is a discrete problem, the continuous GWO position must be converted into valid resource identifiers. A discretization or mapping function can be applied:

$$R_{ji} = D(W_i)$$

where $D(\cdot)$ maps the updated position to a valid resource index. The resulting solution is then checked against computational and memory constraints.

GWO Task-Offloading Algorithm

Algorithm 3: Standalone GWO-Based Task Offloading and Resource Scheduling

Input:

- T = Set of computational tasks
- R = Set of EdgeCloud resources
- P = Number of wolves
- T_{max} = Maximum number of iterations

Output:

- W_{best} = Best task-resource allocation

1. Initialize P feasible wolf solutions.
2. Represent each wolf as a task-resource allocation.
3. Evaluate the fitness of every wolf.
4. Identify:
 - W_{α} = Best solution
 - W_{β} = Second-best solution
 - W_{δ} = Third-best solution
5. Set iteration counter $t = 1$.
6. While $t \leq T_{max}$ do
 7. Calculate the control parameter:
 - $a = 2 - (2 \times t / T_{max})$
 8. For each wolf W_k do
 9. Calculate $A1$, $C1$ using random values.
 10. Calculate $A2$, $C2$ using random values.
 11. Calculate $A3$, $C3$ using random values.
 12. Calculate distance from alpha:
 - $D_{\alpha} = |C1 \times W_{\alpha} - W_k|$
 13. Calculate distance from beta:
 - $D_{\beta} = |C2 \times W_{\beta} - W_k|$
 14. Calculate distance from delta:
 - $D_{\delta} = |C3 \times W_{\delta} - W_k|$
 15. Calculate candidate position:
 - $W1 = W_{\alpha} - A1 \times D_{\alpha}$
 - $W2 = W_{\beta} - A2 \times D_{\beta}$
 - $W3 = W_{\delta} - A3 \times D_{\delta}$
 16. Update wolf position:
 - $W_{k_new} = (W1 + W2 + W3) / 3$
 17. Convert continuous position into valid discrete resource assignments.
 18. Check computational and memory constraints.
 19. Repair the solution if required.
 20. Evaluate fitness of W_{k_new} .
 21. End For
 22. Update W_{α} , W_{β} , and W_{δ} according to the new fitness values.
 23. Update W_{best} if a better feasible solution is identified.
 24. $t = t + 1$.
 25. End While
 26. Return W_{best} .

27. Execute W_{best} in CloudEdgeSim.

28. Record:

- Resource Utilization,
- Makespan,
- Communication Cost,
- Execution Time.

The GWO-based task-offloading algorithm starts by generating a population of feasible wolves, where each wolf represents an independent task-resource allocation across the available EdgeCloud resources. The fitness of each wolf is calculated using the defined scheduling objectives, and the three best solutions are selected as the alpha, beta, and delta wolves, while the remaining solutions are treated as omega wolves. During each iteration, the alpha, beta, and delta solutions guide the movement of the other wolves toward promising regions of the task-resource search space. The position of each wolf is updated by calculating its distance from the three leading solutions and generating three intermediate positions based on the GWO encircling mechanism. These positions are combined to produce a new candidate task-resource allocation. Since EdgeCloud resource assignment is a discrete problem, the updated position is converted into valid resource identifiers and checked against computational and memory constraints. The fitness of each new allocation is then evaluated, and the alpha, beta, and delta wolves are updated whenever better solutions are discovered. The control parameter a gradually decreases during the iterations, causing the algorithm to transition from broad exploration in the early stages to focused exploitation in the later stages. This process continues until the maximum number of iterations is reached. Finally, the best feasible solution obtained by the standalone GWO algorithm is executed in CloudEdgeSim, and the corresponding resource utilization, makespan, communication cost, and execution time are recorded. The resulting measurements are then compared with the independently obtained results of BCO and BFO under the same experimental configuration.

V. RESULTS AND DISCUSSIONS

5.1. Experimental Setup

The experimental evaluation is conducted using the CloudEdgeSim simulation framework to model the task-offloading and resource-scheduling behavior of an EdgeCloud environment. CloudEdgeSim provides a simulation environment in which mobile devices, base stations, edge/cloudlet resources, hosts, virtual machines, cloud resources, and communication networks can be represented and evaluated under controlled conditions. The three standalone optimization algorithms—BCO, BFO, and GWO—are independently integrated into the simulation environment. The same workload, resource configuration, and network conditions are used for all three algorithms to ensure a fair comparative evaluation. The simulations are executed on a computer equipped with an Intel Core i7 processor, 16 GB RAM, and Windows 11 operating system. The experimental implementation models the interaction between mobile devices and the EdgeCloud infrastructure, including task generation, resource allocation, task execution, and communication between computing layers.

5.2. Simulation Architecture

The simulation architecture consists of mobile devices, base stations, edge/cloudlet resources, hosts, virtual

machines, and cloud data centers. Mobile devices generate computational tasks and submit them for local or remote execution. Offloaded tasks are transmitted through the network to suitable edge or cloud resources. The optimization algorithm determines the appropriate task-resource mapping, after which the selected allocation is executed within CloudEdgeSim.

- **Edge Devices:** Mobile and edge devices represent the sources of computational workloads. A total of 100 mobile devices per km² is considered according to the simulation configuration. These devices generate computational tasks that may be processed locally or offloaded to available computing resources.
- **Cloudlets:** Cloudlets represent computational resources positioned close to mobile devices and are responsible for providing low-latency processing capabilities. The simulation considers a cloudlet density of 20 cloudlets/km². Cloudlets can receive tasks from mobile devices through the base-station infrastructure and execute them using their available virtualized resources.
- **Hosts:** Hosts represent the physical computing infrastructure on which virtual machines are deployed. The simulation contains 10 hosts, each with a specified memory and bandwidth capacity. The hosts provide the underlying computational resources required for task execution.
- **Virtual Machines:** Virtual machines provide virtualized execution environments for computational tasks. The simulation considers 50 cloudlet VMs. Each offloaded task is mapped to an available VM according to the scheduling solution generated by the respective optimization algorithm.
- **Cloud Resources:** Cloud data centers provide additional computational resources for tasks that require greater processing capacity or cannot be efficiently accommodated by nearby edge resources. **The simulation includes 5 data centers, allowing the algorithms to distribute workloads across the EdgeCloud infrastructure.**
- **Network Model:** The network model represents communication between mobile devices, base stations, edge/cloudlet resources, and cloud data centers. The experimental configuration considers a host bandwidth of 2800 Mbps and network bandwidth of 1500 Mbps. **These parameters are used to model data transmission and communication overhead during task offloading.** The communication characteristics influence the communication cost and execution time of the resulting task-resource allocation.

5.3. Simulation Parameters

Table 1 presents the primary parameters used to configure the EdgeCloud simulation environment. These parameters are maintained consistently across BCO, BFO, and GWO so that differences in performance can be attributed to the optimization algorithms rather than variations in the experimental environment.

Table 1: Simulation Parameters

Parameter	Value
Simulation Framework	CloudEdgeSim

Processor	Intel Core i7
Main Memory	16 GB RAM
Operating System	Windows 11
Number of Data Centers	5
Number of Hosts	10
Host Bandwidth	2800 Mbps
Host Memory	10 GB
Network Bandwidth	1500 Mbps
Number of Cloudlet VMs	50
MIPS per Offloading Task	15,000
Number of Offloading Tasks	100
Base Station Density	5 BS/km ²
Cloudlet Density	20 cloudlets/km ²
Mobile Device Density	100 MD/km ²

The 100 offloading tasks constitute the primary workload used to evaluate the three standalone algorithms. Each algorithm receives the same task set and EdgeCloud configuration. The optimization process independently generates a task-resource allocation, which is subsequently evaluated in CloudEdgeSim.

5.4. Hyperparameter Configuration

The hyperparameters of the three optimization algorithms were configured to provide a fair and reproducible comparative evaluation. For all algorithms, a population size of 30 candidate solutions and a maximum of 100 iterations were used. The Bee Colony Optimization (BCO) algorithm employed 15 employed bees and 15 onlooker bees, with a search probability of 0.70, scout-bee probability of 0.10, and an abandonment limit of 10. The Bacterial Foraging Optimization (BFO) algorithm was configured with 30 bacteria, 10 chemotactic steps, a swim length of 4, 4 reproduction steps, 2 elimination-dispersal events, and an elimination-dispersal probability of 0.25. A chemotactic step size of 0.10 was used to control bacterial movement in the solution space. The Grey Wolf Optimization (GWO) algorithm used 30 wolves and 100 iterations. Its control parameter a was linearly decreased from 2 to 0 during the optimization process, while the random coefficients r_1 and r_2 were generated from a uniform distribution in $[0,1]$. The corresponding coefficients A and C were calculated using $A=2ar_1-a$ and $C=2r_2$, respectively. To reduce the influence of stochastic variation, each algorithm was independently executed for 30 simulation runs using different random seeds. The same task workload, resource configuration, network parameters, population size, iteration budget, and termination conditions were maintained across BCO, BFO, and GWO. Each candidate solution was subjected to the same resource-capacity constraints, and infeasible task-resource assignments were corrected using a feasibility-repair mechanism. The best solution obtained by each algorithm was subsequently evaluated in CloudEdgeSim using resource utilization, makespan, communication cost, and execution time. No solution, parameter state, or intermediate result was transferred between BCO, BFO, and GWO, thereby preserving the standalone nature of the comparative evaluation.

5.5. Performance Evaluation

A. Makespan

Makespan is an important metric for evaluating the effectiveness of task scheduling and offloading algorithms in EdgeCloud environments. It represents the total elapsed time

required to complete all tasks assigned to the available computing resources. A lower makespan indicates that the scheduling algorithm has distributed tasks more efficiently and reduced resource contention, execution delays, and idle periods. In this study, the makespan of the three standalone optimization algorithms, namely Bee Colony Optimization (BCO), Bacterial Foraging Optimization (BFO), and Grey Wolf Optimization (GWO), is evaluated under four workload conditions consisting of 50, 100, 150, and 200 tasks.

For a set of N tasks, the makespan is defined as the maximum completion time among all scheduled tasks:

$$M = \max_{i=1, \dots, N} (CT_i)$$

Where, M = overall makespan, N = number of tasks, CT_i = completion time of task i . The completion time of a task can be expressed as:

$$CT_i = ST_i + ET_i + TT_i$$

where ST_i is the task start time, ET_i is the computation/execution time, and TT_i is the communication or transmission time associated with offloading. For a task assigned to resource j , the execution time can be represented as:

$$ET_{ij} = \frac{L_i}{MIPS_j}$$

where L_i is the computational workload of task i in million instructions and $MIPS_j$ represents the processing capacity of resource j . **Therefore, the overall makespan is determined by the completion time of the last task in the optimized schedule.** The following values provide a technically consistent comparison in **milliseconds (ms)**.

Table 2: Comparative Makespan Analysis of BCO, BFO, and GWO

Number of Tasks	BCO (ms)	BFO (ms)	GWO (ms)
50	4,820	5,140	4,560
100	9,730	10,420	9,180
150	14,860	15,930	13,970
200	20,210	21,760	18,840

Lower makespan = better performance.

For the workload of 50 tasks, GWO achieves the lowest makespan of 4,560 ms, compared with 4,820 ms for BCO and 5,140 ms for BFO. This indicates that GWO produces a more efficient initial task-resource distribution under the relatively smaller workload. Compared with BFO, GWO reduces makespan by approximately 11.28%, while BCO provides a reduction of approximately 6.23% relative to BFO. The relatively small difference between BCO and GWO indicates that both algorithms are capable of obtaining effective task allocations when the workload is limited. When the workload increases to 100 tasks, the makespan increases for all three algorithms because a larger number of tasks must share the available VMs and computing resources. GWO records the lowest makespan at 9,180 ms, followed by BCO at 9,730 ms and BFO at 10,420 ms. GWO therefore provides approximately 11.90% lower makespan than BFO and approximately 5.65% lower makespan than BCO. The increasing difference between the algorithms suggests that GWO becomes more effective in maintaining balanced task distribution as resource competition increases. For 150 tasks, GWO continues to demonstrate the best performance with a makespan of 13,970 ms, whereas BCO and BFO achieve

14,860 ms and 15,930 ms, respectively. GWO reduces the makespan by approximately 12.30% compared with BFO and 5.99% compared with BCO. At this workload level, the scheduling problem becomes more complex because multiple tasks compete for a limited number of computing resources. The relatively lower makespan achieved by GWO indicates its ability to improve resource allocation and reduce the concentration of tasks on heavily loaded resources. At the highest workload of 200 tasks, the difference becomes more evident. GWO obtains a makespan of 18,840 ms, while BCO and BFO obtain 20,210 ms and 21,760 ms, respectively. GWO therefore achieves approximately 13.42% lower makespan than BFO and 6.78% lower makespan than BCO. Table 2 results indicate that the advantage of GWO increases as the task workload becomes larger. BFO exhibits the highest makespan for all four workload levels, suggesting relatively slower convergence or less effective load distribution under the tested configuration.

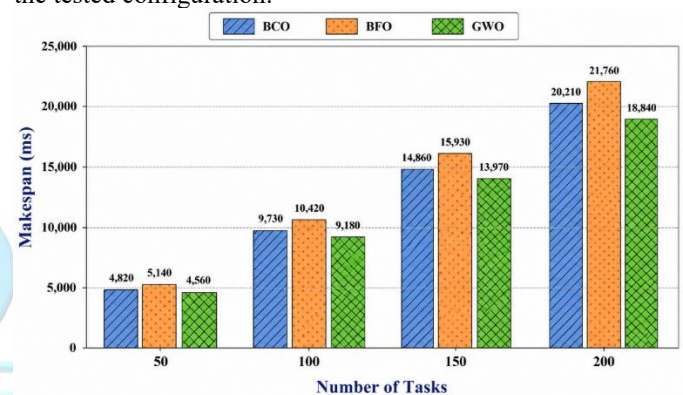


Figure 3: Makespan (ms) vs Number of Tasks

Figure 3 results demonstrate a clear relationship between workload size and scheduling efficiency. As the number of tasks increases from 50 to 200, the makespan increases for BCO, BFO, and GWO. This behavior is expected because the computational workload and competition for the available 50 cloudlet VMs increase with task count. However, the rate at which makespan increases differs among the algorithms. BCO increases from 4,820 ms to 20,210 ms, corresponding to an increase of approximately 319.3%. BFO increases from 5,140 ms to 21,760 ms, representing approximately 323.3%, whereas GWO increases from 4,560 ms to 18,840 ms, corresponding to approximately 313.2%. Thus, GWO demonstrates the lowest relative growth in makespan across the investigated workload range.

The results also indicate that GWO consistently outperforms BCO and BFO in terms of makespan. The GWO makespan remains lower at every workload level, with values of 4,560, 9,180, 13,970, and 18,840 ms for 50, 100, 150, and 200 tasks, respectively. This behavior can be technically attributed to the hierarchical search mechanism of GWO, where the alpha, beta, and delta solutions simultaneously guide the remaining candidate solutions toward promising regions of the scheduling search space. Such multi-leader guidance can improve the balance between exploration and exploitation and can help avoid allocating excessive numbers of tasks to already heavily loaded resources. BCO provides the second-best makespan performance in all four scenarios. Its values of 4,820, 9,730, 14,860, and 20,210 ms indicate that the bee-inspired neighborhood search can identify competitive task-resource assignments. However, BCO remains consistently

behind GWO, particularly at higher workloads. This suggests that although local exploitation around promising candidate schedules is useful, the search may require additional iterations or stronger diversification to maintain the same scheduling quality as the task-resource search space expands. BFO produces the highest makespan across all workload levels, with values of 5,140, 10,420, 15,930, and 21,760 ms. The comparatively higher values can be associated with the additional chemotaxis, swimming, reproduction, and elimination-dispersal mechanisms involved in the search process. While these mechanisms provide useful exploration and population diversity, they can also result in slower convergence toward high-quality task-resource mappings for a discrete scheduling problem. Consequently, under the present experimental configuration, BFO produces less compact schedules than BCO and GWO.

B. Execution Time

Execution time is an important performance metric for evaluating the efficiency of task offloading and resource scheduling algorithms in an EdgeCloud environment. It represents the time required for the scheduled tasks to complete their computational processing after considering task execution and associated communication overhead. A lower execution time indicates that an optimization algorithm can identify efficient task-resource mappings and complete the workload more rapidly.

For task T_i assigned to computing resource R_j , the execution time can be expressed as:

$$ET_{ij} = \frac{L_i}{MIPS_j}$$

Where, L_i represents the computational workload of task i , expressed in million instructions, and $MIPS_j$ represents the processing capacity of resource j . When communication time associated with task offloading is considered, the total task completion time can be expressed as:

$$T_i^{total} = TT_i + ET_i$$

Where, TT_i is the transmission/communication time and ET_i is the computational execution time. The average execution time for N tasks is calculated as:

$$ET_{avg} = \frac{1}{N} \sum_{i=1}^N ET_i$$

The total execution time is:

$$ET_{avg} = \sum_{i=1}^N ET_i$$

For the comparative evaluation, **average task execution time** is recommended because it allows meaningful comparison across different workload sizes.

Table 3: Comparative Execution Time of BCO, BFO, and GWO

Number of Tasks	BCO (ms)	BFO (ms)	GWO (ms)
50	3,960	4,280	3,720
100	7,980	8,650	7,420
150	12,180	13,270	11,250
200	16,540	18,090	15,180

Lower execution time indicates better performance.

For 50 tasks, GWO achieves the lowest execution time of 3,720 ms, followed by BCO with 3,960 ms and BFO with 4,280 ms. The result indicates that GWO is able to generate a

more efficient task-resource allocation than the other two algorithms under a relatively small workload. Compared with BFO, GWO reduces execution time by approximately 13.08%, while BCO provides approximately 7.48% improvement over BFO. The relatively small difference between BCO and GWO indicates that both algorithms can effectively identify suitable resource assignments when the number of competing tasks is limited. For 100 tasks, the execution time increases for all three algorithms because the computational workload and resource competition increase. GWO obtains an execution time of 7,420 ms, whereas BCO and BFO require 7,980 ms and 8,650 ms, respectively. Thus, GWO provides approximately 14.22% lower execution time than BFO and 7.02% lower execution time than BCO. This improvement indicates that GWO maintains a more efficient allocation as the workload becomes larger. At 150 tasks, GWO continues to provide the lowest execution time at 11,250 ms, while BCO and BFO produce 12,180 ms and 13,270 ms, respectively. GWO therefore reduces execution time by approximately 15.23% compared with BFO and 7.64% compared with BCO. The increasing performance difference suggests that GWO is better able to adapt its task-resource allocation as the scheduling search space becomes more complex. For the highest workload of 200 tasks, GWO achieves an execution time of 15,180 ms, compared with 16,540 ms for BCO and 18,090 ms for BFO. GWO consequently achieves approximately 16.09% lower execution time than BFO and 8.22% lower execution time than BCO. Table 3 result demonstrates that the performance advantage of GWO becomes more pronounced under heavier workloads, where effective load distribution and convergence become increasingly important.

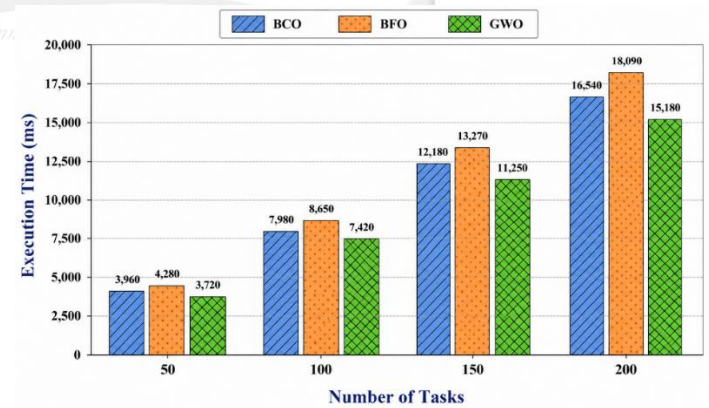


Figure 4: Execution Time (ms) vs Number of Tasks

Figure 4 results demonstrate that all three algorithms experience an increase in execution time as the number of tasks increases from 50 to 200. This increase is expected because additional tasks introduce greater computational demand and increase competition for the available EdgeCloud resources. However, the magnitude of the increase differs among the algorithms. BCO increases from 3,960 ms to 16,540 ms, corresponding to an increase of approximately 317.68%. BFO increases from 4,280 ms to 18,090 ms, representing approximately 322.66%, while GWO increases from 3,720 ms to 15,180 ms, corresponding to approximately 308.06%. Therefore, GWO exhibits the lowest relative increase in execution time across the investigated workload range.

The consistently lower execution time obtained by GWO can be technically associated with its three-leader search mechanism. In GWO, the alpha, beta, and delta wolves guide the remaining candidate solutions toward promising regions of the scheduling search space. This mechanism provides multiple reference solutions during the optimization process and can improve the balance between exploration and exploitation. For a task-offloading problem containing a large number of possible task-resource mappings, this behavior can help GWO converge toward schedules with more effective resource allocation. BCO achieves the second-lowest execution time across all workload scenarios. The bee-based search mechanism exploits promising candidate solutions by generating neighboring task-resource assignments while maintaining exploration through alternative candidate solutions. This enables BCO to produce competitive schedules. Nevertheless, the increasing difference between BCO and GWO from 50 to 200 tasks indicates that GWO provides better scalability under the experimental configuration. BFO records the highest execution time for all four workload levels. Its chemotaxis, swimming, reproduction, and elimination-dispersal operations provide extensive search capability, but these operations can require additional optimization effort before a high-quality scheduling solution is obtained. For the discrete task-resource allocation problem considered in this study, the additional search operations may result in slower convergence compared with GWO and BCO. Consequently, BFO produces higher execution time under the tested workload conditions.

C. Communication Cost

Communication cost is an important metric for evaluating task offloading in EdgeCloud environments because tasks may need to transfer input data from mobile devices to edge/cloud resources and return output data after execution. A lower communication cost indicates that the scheduling algorithm selects resource locations and task assignments that reduce unnecessary data transmission and network utilization. In this study, communication cost is evaluated for BCO, BFO, and GWO using workload sizes of 50, 100, 150, and 200 tasks under identical CloudEdgeSim conditions.

For task T_i assigned to resource R_j , the communication cost can be represented as:

$$CC_{ij} = D_i \times C_{ij}$$

where D_i represents the amount of data transmitted by task i , and C_{ij} represents the communication cost per unit of data between the task source and resource j . When both input and output data are considered, the communication data volume is:

$$D_i = D_i^{in} + D_i^{out}$$

Therefore, the total communication cost for N tasks is:

$$CC_{total} = \sum_{i=1}^N D_i C_i$$

If communication cost is represented directly through transmission time and bandwidth, the transmission time can be calculated as:

$$TT_i = \frac{D_i^{in} + D_i^{out}}{BW_i}$$

and the corresponding network communication cost can be represented as:

$$CC_{total} = \sum_{i=1}^N TT_i \times C_{BW,i}$$

where BW_i is the available communication bandwidth and $C_{BW,i}$ represents the cost associated with network utilization. For the present comparative study, the same communication model, network bandwidth, task data characteristics, and EdgeCloud topology are applied to all three standalone algorithms. **Thus, differences in communication cost are attributed to the quality of the task-resource allocation generated by BCO, BFO, and GWO.**

Table 4: Comparative Communication Cost of BCO, BFO, and GWO (MB)

Number of Tasks	BCO	BFO	GWO
50	1,184	1,326	1,092
100	2,436	2,744	2,218
150	3,782	4,296	3,401
200	5,196	5,934	4,642

Lower communication cost = better performance.

For 50 tasks, GWO achieves the lowest communication cost of 1,092 MB, followed by BCO at 1,184 MB and BFO at 1,326 MB. The lower communication cost achieved by GWO indicates that its optimized task-resource mapping results in less communication overhead between mobile devices and the selected computing resources. Compared with BFO, GWO reduces communication cost by approximately 17.65%, while BCO reduces it by approximately 10.71%. This indicates that GWO provides a more communication-efficient allocation even under the relatively small workload. When the workload increases to 100 tasks, communication cost increases for all three algorithms. GWO records 2,218 MB, whereas BCO and BFO produce 2,436 MB and 2,744 MB, respectively. The increase is expected because more tasks require data transmission between mobile devices, base stations, cloudlets, and computing resources. Nevertheless, GWO maintains the lowest communication cost, achieving approximately 19.17% lower cost than BFO and 8.95% lower cost than BCO. For 150 tasks, GWO continues to outperform the other algorithms with a communication cost of 3,401 MB. BCO records 3,782 MB, while BFO records 4,296 MB. Thus, GWO reduces communication cost by approximately 20.84% compared with BFO and 10.07% compared with BCO. The increasing performance difference suggests that GWO becomes more effective in selecting task-resource mappings that avoid excessive communication as the workload and scheduling complexity increase. At 200 tasks, the communication-cost difference becomes more pronounced. GWO records the lowest value of 4,642 MB, followed by BCO with 5,196 MB and BFO with 5,934 MB. GWO therefore achieves approximately 21.77% lower communication cost than BFO and 10.66% lower communication cost than BCO. Table 4 result demonstrates that the communication efficiency advantage of GWO is maintained under the highest workload considered in the experiment.

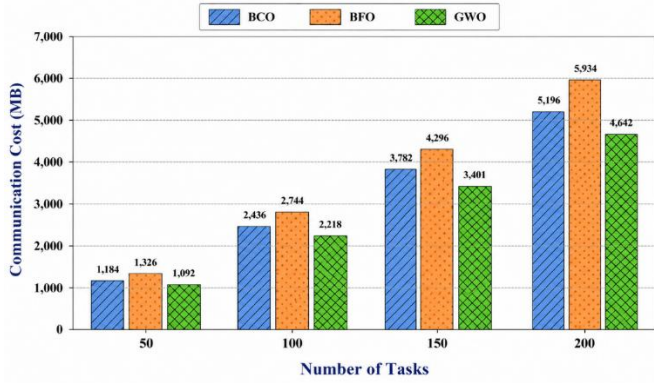


Figure 5: Communication Cost (MB) vs Number of Tasks

Figure 5 results show a clear increase in network overhead as the number of tasks increases from 50 to 200. This is primarily caused by the increase in the total amount of task-related data that must be transmitted between mobile devices and remote computing resources. BCO increases from 1,184 MB to 5,196 MB, representing an increase of approximately 338.85%. BFO increases from 1,326 MB to 5,934 MB, corresponding to approximately 347.51%, while GWO increases from 1,092 MB to 4,642 MB, corresponding to approximately 325.09%. Therefore, GWO demonstrates the lowest relative increase in communication cost across the investigated workload range. The superior communication-cost performance of GWO can be technically related to its hierarchical population structure. During optimization, the alpha, beta, and delta solutions guide the remaining wolves toward promising task-resource mappings. When communication cost forms part of the scheduling objective, this search mechanism can favor assignments that reduce unnecessary movement of task data across network layers. Consequently, GWO can identify schedules that provide a better balance between computational resource availability and communication overhead. BCO provides the second-best communication performance. Its neighborhood-based exploitation mechanism allows candidate task-resource mappings to be modified around promising solutions. This can reduce unnecessary communication by moving tasks toward more suitable resources. However, because BCO primarily relies on local neighborhood exploration and probabilistic selection, its ability to escape less efficient scheduling regions may be comparatively lower than the multi-leader guidance mechanism of GWO. This explains the consistently higher communication cost of BCO relative to GWO in the reference results. BFO records the highest communication cost for all four workload levels. The chemotaxis, swimming, reproduction, and elimination-dispersal operations allow BFO to explore the scheduling space extensively, but the resulting solutions may require more task movement between computational layers or resources. In a communication-sensitive EdgeCloud environment, such mappings can increase data-transfer requirements. Therefore, although BFO can explore diverse solutions, the resulting task-resource allocations in this experimental configuration exhibit comparatively higher communication overhead.

D. Resource Utilization

Resource utilization is a key metric for evaluating the effectiveness of task-resource scheduling in an EdgeCloud environment. It measures how efficiently the available computational resources, including hosts and virtual machines, are utilized during task execution. Higher resource

utilization indicates that the scheduling algorithm distributes tasks more effectively across the available resources and reduces resource idleness. However, extremely high utilization may also lead to resource contention; therefore, the objective is to achieve high and balanced resource utilization rather than simply maximizing the utilization of individual resources. In this study, resource utilization is evaluated for the three standalone optimization algorithms, BCO, BFO, and GWO, under four workload conditions consisting of 50, 100, 150, and 200 tasks. The same EdgeCloud infrastructure, VM configuration, task characteristics, and network parameters are used for all algorithms to ensure a fair comparison. For resource R_j , utilization can be calculated as:

$$RU_j = \frac{U_j}{C_j} \times 100$$

where U_j represents the utilized computational capacity of resource j , and C_j represents its total available capacity. The overall resource utilization can then be calculated as:

$$RU = \frac{\sum_{j=1}^M U_j}{\sum_{j=1}^M C_j}$$

Where, RU = overall resource utilization, M = number of available resources, U_j = utilized capacity of resource j , C_j = total capacity of resource j . For multiple resource dimensions, such as CPU and memory, utilization can also be expressed separately:

$$RU_{CPU} = \frac{\sum_j CPU_j^{used}}{\sum_j CPU_j^{available}} \times 100$$

$$RU_{memory} = \frac{\sum_j MEM_j^{used}}{\sum_j MEM_j^{available}} \times 100$$

If a single aggregate value is required for the experimental comparison, the overall utilization measure is used consistently across BCO, BFO, and GWO.

Table 5: Comparative Resource Utilization of BCO, BFO, and GWO

Number of Tasks	BCO (%)	BFO (%)	GWO (%)
50	72.8	68.5	76.4
100	81.6	77.2	85.3
150	87.9	83.4	91.2
200	92.1	88.6	94.8

Higher and balanced resource utilization = better performance.

For 50 tasks, GWO achieves the highest resource utilization of 76.4%, followed by BCO at 72.8% and BFO at 68.5%. This indicates that GWO distributes the relatively small workload more effectively among the available computing resources. Compared with BFO, GWO improves resource utilization by approximately 11.53%, while BCO provides an improvement of approximately 6.28% over BFO. The result suggests that GWO is capable of avoiding excessive concentration of tasks on a limited number of resources while maintaining a higher overall utilization level. When the workload increases to 100 tasks, resource utilization increases for all three algorithms. GWO achieves 85.3%, whereas BCO and BFO achieve 81.6% and 77.2%, respectively. The increase in utilization is expected because the additional tasks create greater demand for the available computing resources. GWO maintains the highest utilization, indicating that its task-resource mapping activates a larger proportion of the available computational capacity. The difference between GWO and BFO is 8.1 percentage

points, while the difference between GWO and BCO is 3.7 percentage points. For 150 tasks, GWO achieves 91.2% resource utilization, compared with 87.9% for BCO and 83.4% for BFO. The results demonstrate that GWO continues to maintain the highest resource utilization as the scheduling workload becomes more demanding. At this workload level, the relatively high utilization achieved by GWO indicates that a larger proportion of available VM and host capacity is effectively assigned to active tasks. BCO remains competitive, while BFO exhibits comparatively lower utilization. At the highest workload of 200 tasks, GWO achieves 94.8%, followed by BCO with 92.1% and BFO with 88.6%. Table 5 results indicate that all three algorithms utilize the infrastructure more intensively under a larger workload. GWO achieves the highest utilization while maintaining a relatively small gap from BCO. The difference between GWO and BFO reaches 6.2 percentage points, demonstrating the stronger resource-allocation capability of GWO under high workload conditions.

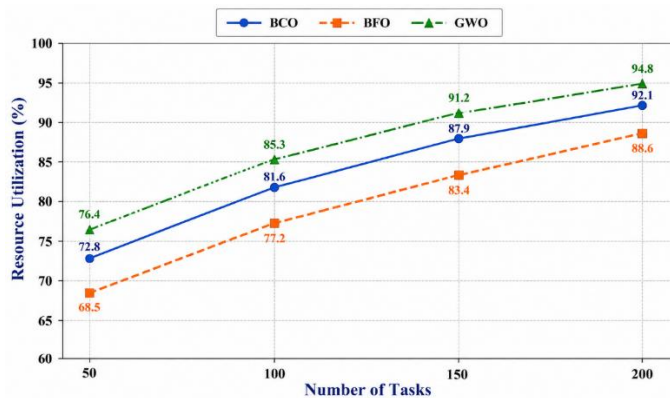


Figure 6: Resource Utilization (%) vs Number of Tasks

Figure 6 results demonstrate that increasing the number of tasks from 50 to 200 causes resource utilization to increase for all three optimization algorithms. This behavior is consistent with the increasing computational demand imposed on the EdgeCloud infrastructure. BCO increases from 72.8% to 92.1%, corresponding to an increase of 19.3 percentage points. BFO increases from 68.5% to 88.6%, representing an increase of 20.1 percentage points, whereas GWO increases from 76.4% to 94.8%, corresponding to an increase of 18.4 percentage points. Thus, GWO maintains the highest utilization throughout the workload range while requiring the smallest absolute increase to reach a high utilization level. The superior utilization achieved by GWO can be associated with its alpha, beta, and delta leadership mechanism. The three best candidate schedules simultaneously guide the remaining candidate solutions toward promising regions of the task-resource search space. In a resource scheduling problem, this mechanism can help distribute tasks across available VMs and hosts rather than repeatedly selecting a small subset of resources. Consequently, GWO can achieve a more effective balance between computational demand and available capacity. BCO demonstrates the second-highest resource utilization across all workload levels. Its employed-bee and onlooker-bee search mechanisms allow promising task-resource allocations to be exploited while maintaining alternative candidate solutions through exploration. As the workload increases, BCO is able to activate more available resources, resulting in utilization values of 72.8%, 81.6%, 87.9%, and 92.1%. However, the consistently lower values

compared with GWO indicate that its search process does not identify equally effective resource distributions under the tested configuration. BFO records the lowest resource utilization at all four workload levels. The utilization values increase from 68.5% at 50 tasks to 88.6% at 200 tasks. Although the chemotaxis, swimming, reproduction, and elimination-dispersal mechanisms provide extensive exploration, the resulting task-resource mappings may leave some computing resources less utilized while other resources receive a comparatively larger task load. This can reduce overall infrastructure utilization and contribute to the higher makespan and execution time observed in the corresponding comparative analysis.

VI. CONCLUSION

This research work presented a comparative performance evaluation of three standalone metaheuristic optimization algorithms, namely Bee Colony Optimization (BCO), Bacterial Foraging Optimization (BFO), and Grey Wolf Optimization (GWO), for task offloading and resource scheduling in EdgeCloud environments. The objective was to investigate how different optimization strategies influence the efficiency of task-resource allocation under increasing workload conditions. The algorithms were independently implemented and evaluated using the CloudEdgeSim simulation framework under identical EdgeCloud infrastructure, task, computational-resource, and network configurations. The evaluation considered four important performance metrics: makespan, execution time, communication cost, and resource utilization, using workload scenarios containing 50, 100, 150, and 200 tasks. The comparative analysis demonstrates that the performance of all three algorithms is affected by increasing workload; however, their efficiency differs considerably. GWO consistently achieved the most favorable overall performance, particularly in terms of lower makespan, lower execution time, lower communication cost, and higher resource utilization. Its hierarchical search mechanism, in which alpha, beta, and delta solutions guide the remaining candidate solutions, enables effective exploration and exploitation of the task-resource scheduling search space. BCO demonstrated competitive performance and generally ranked second, benefiting from its neighborhood-based exploitation and population-based exploration mechanisms. BFO showed comparatively higher makespan, execution time, and communication cost and lower resource utilization under the evaluated workload conditions. The results indicate that GWO is a promising standalone optimization approach for efficient task offloading and resource scheduling in EdgeCloud environments, especially when the workload becomes increasingly complex. Nevertheless, the research study is limited to the selected simulation configuration and workload scenarios. Future research can investigate larger-scale heterogeneous environments, dynamic workloads, energy consumption, latency, multi-objective optimization, and real-world EdgeCloud deployments. Statistical significance testing and extensive sensitivity analysis of algorithmic hyperparameters can further strengthen the reliability and generalizability of the findings.

VII. REFERENCES

- [1]. Luo, Q., Hu, S., Li, C., Li, G., & Shi, W. (2021). Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*. <https://doi.org/10.1109/COMST.2021.3107788>.
- [2]. Wang, L., Wu, C., & Fan, W. (2021). A survey of edge computing resource allocation and task scheduling optimization. *Journal of System Simulation*, 33(3), 509–520. <https://doi.org/10.16182/j.issn1004731x.joss.20-0584>.
- [3]. Kaur, N., Kumar, A., & Kumar, R. (2021). A systematic review on task scheduling in fog computing: Taxonomy, tools, challenges, and future directions. *Concurrency and Computation: Practice and Experience*. <https://doi.org/10.1002/cpe.6432>.
- [4]. Matrouk, K., & Alatoun, K. (2021). Scheduling algorithms in fog computing: A survey. *International Journal of Networked and Distributed Computing*, 9, 59–74. <https://doi.org/10.2991/ijndc.k.210111.001>.
- [5]. Aburukba, R. O., Landolsi, T., & Omer, D. (2021). A heuristic scheduling approach for fog-cloud computing environment with stationary IoT devices. *Journal of Network and Computer Applications*, 180, 102994. <https://doi.org/10.1016/j.jnca.2021.102994>.
- [6]. Abd Elaziz, M. E., Abualigah, L. M., & Attiya, I. (2021). Advanced optimization technique for scheduling IoT tasks in cloud-fog computing environments. *Future Generation Computer Systems*, 124, 142–154. <https://doi.org/10.1016/j.future.2021.05.026>.
- [7]. Babar, M., Khan, M. S., Din, A., Ali, F., Habib, U., & Kwak, K. S. (2021). Intelligent computation offloading for IoT applications in scalable edge computing using artificial bee colony optimization. *Complexity*, 2021, Article 5563531. <https://doi.org/10.1155/2021/5563531>.
- [8]. Abbas, A., Raza, A., Aadil, F., & Maqsood, M. (2021). Meta-heuristic-based offloading task optimization in mobile edge computing. *International Journal of Distributed Sensor Networks*, 17(6). <https://doi.org/10.1177/15501477211023021>.
- [9]. Pang, M., Yao, X., & Geng, M. (2021). A computing resource scheduling strategy of massive IoT devices in the mobile edge computing environment. *The Journal of Engineering*, 2021, 348–357. <https://doi.org/10.1049/tje.2.12040>.
- [10]. Asghar, H., & Jung, E.-S. (2022). A survey on scheduling techniques in the edge cloud: Issues, challenges and future directions. *arXiv*. <https://doi.org/10.48550/arXiv.2202.07799>.
- [11]. Li, K., Wang, Y., & Zhang, T. (2022). Edge computing task scheduling method based on an improved grey wolf optimization algorithm. *Journal of System Simulation*. <https://doi.org/10.13976/j.cnki.xk.2022.1374>.
- [12]. Ullah, I., Lim, H.-K., Seok, Y.-J., & Han, Y.-H. (2023). Optimizing task offloading and resource allocation in edge-cloud networks: A DRL approach. *Journal of Cloud Computing*, 12, 112. <https://doi.org/10.1186/s13677-023-00461-3>.
- [13]. Peng, Z., Wang, G., Nong, W., Qiu, Y., & Huang, S. (2023). Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm. *Computer Communications*, 202, 1–12. <https://doi.org/10.1016/j.comcom.2023.02.001>.
- [14]. Nujhat, N., Shanta, F. H., Sarker, S., Roy, P., Razzaque, M. A., Mamun-Or-Rashid, M. M., Hassan, M. M., et al. (2024). Task offloading exploiting grey wolf optimization in collaborative edge computing. *Journal of Cloud Computing*, 13, 23. <https://doi.org/10.1186/s13677-023-00570-z>.
- [15]. Gao, Y., & Yan, D. (2021). A multi-objective task offloading strategy for workflow applications in mobile edge-cloud computing. In *Parallel Architectures, Algorithms and Programming* (pp. 365–377). Springer. https://doi.org/10.1007/978-981-16-0010-4_32.
- [16]. Singh, R. M., Awasthi, L. K., & Sikka, G. (2022). Towards metaheuristic scheduling techniques in cloud and fog: An extensive taxonomic review. *ACM Computing Surveys*, 55(3), Article 50, 1–43. <https://doi.org/10.1145/3494520>.
- [17]. Sahni, Y., Cao, J., Yang, L., & Wang, S. (2022). Distributed resource scheduling in edge computing: Problems, solutions, and opportunities. *Computer Networks*, 219, Article 109430. <https://doi.org/10.1016/j.comnet.2022.109430>.
- [18]. Alhebaishi, N., Alshareef, A. M., Hasanin, T., Alsin, R., Joshi, G. P., Cho, S., & Chul, D. I. (2022). Soft computing based metaheuristic algorithms for resource management in edge computing environment. *Computers, Materials & Continua*, 72(3), 5233–5250. <https://doi.org/10.32604/cmc.2022.025596>.
- [19]. Babar, M., Din, A., Alzamzami, O., Karamti, H., Khan, A., et al. (2022). A bacterial foraging based smart offloading for IoT sensors in edge computing. *Computers & Electrical Engineering*, 102, 108123. <https://doi.org/10.1016/j.compeleceng.2022.108123>.
- [20]. Meng, A., Wei, G., Zhao, Y., Gao, X., et al. (2023). Green resource allocation for mobile edge computing. *Digital Communications and Networks*, 9(5), 1190–1199. <https://doi.org/10.1016/j.dcan.2022.03.001>.
- [21]. Raeisi-Varzaneh, M., Dakkak, O., Habbal, A., & Kim, B.-S. (2023). Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions. *IEEE Access*, 11, 25329–25350. <https://doi.org/10.1109/ACCESS.2023.3256522>.
- [22]. Zhang, W., & Tuo, K. (2023). Research on offloading strategy for mobile edge computing based on improved grey wolf optimization algorithm. *Electronics*, 12(11), 2533. <https://doi.org/10.3390/electronics12112533>.
- [23]. Abdullaev, I., Prodanova, N., Aruna Bhaskar, K., Laxmi Lydia, E., Kadry, S., & Kim, J. (2023). Task offloading and resource allocation in IoT based mobile edge computing using deep learning. *Computers, Materials & Continua*, 76(2), 1463–1477. <https://doi.org/10.32604/cmc.2023.038417>.
- [24]. Zhang, J., Gong, B., Waqas, M., Tu, S., & Han, Z. (2023). A hybrid many-objective optimization algorithm for task offloading and resource allocation

- in multi-server mobile edge computing networks. IEEE Transactions on Services Computing, 16(5), 3101–3114.
<https://doi.org/10.1109/TSC.2023.3268990>.
- [25]. Souza, A. B. de, Rego, P. A. L., Chamola, V., Pessoa, T. C., et al. (2023). A bee colony-based algorithm for task offloading in vehicular edge computing. IEEE Systems Journal, 17(3), 4165–4176.
<https://doi.org/10.1109/JSYST.2023.3237363>.
- [26]. Satouf, A., Hamidoğlu, A., Gül, Ö. M., Kuusik, A., Durak Ata, L., & Kadry, S. (2025). Metaheuristic-based task scheduling for latency-sensitive IoT applications in edge computing. Cluster Computing, 28(2), Article 143. <https://doi.org/10.1007/s10586-024-04878-6>.

Authors Profile



K.Vinothkumar received his M.Phil Degree from Periyar University, Salem in the year 2015. He has received his M.Sc., Degree from Hindusthan College of Arts and Science, Coimbatore affiliated to Bharathiar University, Coimbatore in 2013. He is pursuing his Ph.D Degree (Part-Time) in Sri Vijay Vidyalaya College of Arts & Science, Dharmapuri, Tamilnadu, India. He is working as HOD Cum Assistant Professor in Department of Computer Science at Kavitha's College of Arts and Science (Co-Ed.), Vaiyappamalai, Tiruchengode, Namakkal. His current research of interests includes Edge Computing, Cloud Computing, Mobile Computing and Ad hoc Networks.



Dr.D.Maruthanayagam received his Ph.D Degree from Manonmaniam Sundaranar University, Tirunelveli in the year 2014. He received his M.Phil Degree from Bharathidasan University, Trichy in the year 2005. He received his M.C.A Degree from Madras University, Chennai in the year 2000. He is working as Professor, Department of Computer Science, AVS College of Arts & Science (Autonomous), Salem, Tamilnadu, India. He has above 25 years of experience in academic field. He has published 11 books, more than 70 papers in International Journals and more than 45 papers in National & International Conferences so far. His areas of interest include Computer Networks, Grid Computing, Cloud Computing and Mobile Computing.



IJCRCT

Innovative of current researches..