Task Scheduling Problem in Fog Computing Environment with improved Memetic algorithm

Dang Van Thang, Volkov Artem, Ammar Muthanna, Olga Vorozheikina, Andrey Koucheryavy
The Bonch-Bruevich Saint-Petersburg State University of Telecommunications, St. Petersburg, 193232, Russian Federation

ABSTRACT

The task scheduling problem in fog computing is one of the key challenges in the development of fog computing within next-generation communication networks. Addressing this challenge requires balancing processing performance with resource constraints while meeting network conditions. Given the distributed and heterogeneous nature, as well as the dynamic topology, optimally allocating tasks to fog nodes is a complex issue. To contribute to solving this problem, we propose a task scheduling method based on an improved Memetic algorithm. The proposed method leverages the strengths of evolutionary algorithms and local search, while incorporating a task restructuring mechanism, to enhance allocation efficiency and task processing in the fog computing environment. Simulation experiments demonstrate that the proposed method outperforms genetic algorithms, Round-Robin, Greedy methods, and the Ant Colony Optimization algorithm in terms of efficiency. This study provides a fresh, simpler approach that aligns with network conditions while still achieving the desired performance.

KEYWORDS: *6G, fog computing, task allocation, memetic algorithm, task restructuring mechanism, next-generation networks.*

1. INTRODUCTION

The explosion of the Internet of Things (IoT) has generated a massive amount of data and an urgent need to process it. Meanwhile, the traditional cloud computing model is increasingly unable to meet the growing demands for bandwidth, latency, and robust security from remote edge devices. Fog computing has emerged as a highly innovative solution to address this issue. However, a reality remains: fog nodes do not always meet real-time dynamic demands due to limitations in computational resources. The fog computing environment faces several significant constraints. First, fog nodes often have limited computation time due to their dynamic nature, with the ability to join or leave a cluster unpredictably. Second, the computational resources of fog nodes, including CPU, RAM, and bandwidth, are typically more restricted compared to cloud data centers, posing challenges in handling complex tasks or multiple tasks simultaneously. Finally, the structure of a fog cluster lacks stability because nodes can unexpectedly disconnect or leave, affecting task allocation and execution processes. These limitations require scheduling methods that are not only optimized for performance but also flexible and quick to adapt to environmental changes.

To tackle this issue, various task scheduling methods for the fog computing environment have been researched and proven effective to varying degrees. Reality shows that, with their limited computational capacity, applying advanced superior algorithms such as reinforcement learning, deep learning, or other methods to the fog computing environment remains challenging. Therefore, a suitable scheduling algorithm that optimizes task allocation, ensures cost-effectiveness and energy efficiency, and meets other imposed requirements is a highly important and pressing matter.

Within the scope of this paper, we propose a task scheduling method for Fog Computing based on the application of the Memetic algorithm and a task restructuring mechanism. With a simple structure yet integrating evolutionary methods and local search, along with a task schedule restructuring approach, the proposed method has demonstrated its effectiveness compared to several other methods such as Greedy, GA, Round Robin, and ACO. The results of the method are described and evaluated in detail in the experimental and simulation section later in the paper. The application of the proposed

https://doi.org/10.54389/OVFU5465

method in this paper contributes to addressing the ongoing challenge of task scheduling in dynamic and resource-constrained environments like Fog Computing.

This study is divided into 5 sections, in which Sections 1 and 2 introduce the problem and the motivation for the research. In Sectiomethod andsent the system structure of Fog Computing, the proposed method, and explain related issues. Experimental simulation, data collection, and result evaluation are presented in Section 4. Finally, the conclusion and future research directions are provided.

2. RESEARCH MOTIVATION

Fog computing emerges as a significant advancement in processing data from the Internet of Things (IoT) network, particularly in applications requiring low latency and optimal bandwidth, such as telepresence services, telecommunication-type holography, and sensory Internet [1].

Researched and developed as a solution to overcome critical limitations of the traditional cloud computing model, Fog computing plays a substantial role in managing, processing, and analyzing data, while also meeting certain requirements such as latency, optimal bandwidth usage, and real-time responsiveness. Additionally, Fog computing brings computational and data processing tasks to more remote locations where edge devices are situated, which are generating an enormous volume of data. This helps save transmission costs and mitigate network congestion [2]. However, Fog nodes are facing significant challenges in their development due to limitations in computational resources and the capacity to handle tasks compared to the cloud [3].

One of the greatest challenges Fog computing encounters is the task optimization problem under conditions constrained by computational capacity. Therefore, efficient techniques for task scheduling and resource management are needed to enhance the operational efficiency of Fog nodes [4]. Several methods have been proposed to address this issue, including algorithms based on search, optimization, and machine learning approaches. Heuristic algorithms such as the Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Simulated Annealing (SA) are commonly applied to solve the task scheduling problem in the Fog computing environment. However, a prominent common limitation of these algorithms is the difficulty in achieving a globally optimal solution. The explanation for this limitation is that these algorithms often get trapped at local optimization bottlenecks, particularly in the highly dynamic Fog computing environment. Furthermore, as mentioned above, the constraints of Fog nodes lead to difficulties in deploying advanced algorithms such as deep learning and reinforcement learning, which require substantial computational resources [5].

Within the scope of this research, to address the limitations mentioned above in the Fog computing structure, we propose a hybrid Memetic Algorithm (MA) combined with a task restructuring mechanism, as a feasible solution. The Memetic Algorithm is a hybrid metaheuristic that integrates local search processes within the framework of evolutionary algorithms [6], [7]. This class of hybrid algorithms leverages the strengths of both local and global search strategies, yielding better solutions. With the integrated task restructuring mechanism, the system can quickly detect unavailable or overloaded nodes and transfer the affected tasks to other nodes within the cluster. In this way, the restructuring mechanism not only ensures the continuity of task execution but also optimizes resource utilization in a dynamic environment. This is particularly crucial in real-time applications such as telepresence, sensory Internet, and holography, where latency and continuity are key factors. The operational mechanism of the proposed method is described in the subsequent section of this study, and additionally, the effectiveness of the method is demonstrated in the simulation and result evaluation section.

3. SYSTEM ARCHITECTURE AND PROPOSED METHOD

3.1 Architecture of Fog Computing System

First introduced by Cisco, Fog Computing is a technological innovation encompassing AI (Artificial Intelligence) and IoT [8]. Many enterprises and organizations in the industry have begun deploying fog computing ecosystems, including communication providers such as Cisco and Huawei, as well as numerous cloud computing and IoT enterprises. The concept of Fog Computing is understood

as a method of transferring part of the data center operations to the network edge and to the level of user devices. It provides a smaller amount of storage, processing, and network resources in the form of sharing between the CC data center and the end device. Fog computing constitutes distributed dynamic architectures, comprising multiple devices and small data centers located close to the sources of data generation, such as sensors, mobile phones, and portable devices.

In a hierarchical architecture, fog computing plays the role of an intermediary, positioned between the end layer and the remote cloud computing layer to ensure the continuity of service provision along the device-fog-cloud path [9]. In this context, the hierarchical model of the fog computing architecture is illustrated in Fig. 1.

End device layer: The end device layer mainly consists of sensor devices and various types of end devices or sensor devices responsible for processing during the data collection process.

Fog computing layer: The fog computing layer is located at a position between the end device layer and the cloud computing layer. Fog nodes deployed at the network edge primarily provide services for latency-sensitive tasks. For some common computing devices, such as smart sensors, smart processing devices, smart multimedia devices, they can also be used as fog nodes. Fog nodes have certain computational capabilities, communication capabilities, and storage capabilities, and they operate between the cloud and end devices. Fog computing has overcome the high latency of cloud computing, thus the quality of service of real-time applications can be guaranteed. However, the capabilities of fog nodes are ultimately limited; fog nodes also exhibit heterogeneity and strong dynamics when facing a large amount of real-time processing data, therefore, how to analyze data quickly and efficiently is the primary goal of fog computing.

Cloud computing layer: The cloud computing layer contains the cloud data center, cloud storage, cloud computing devices, providing remote services for smart production lines. High-performance computing devices and large-capacity storage devices can transmit, compute, and store all kinds of massive data from end devices through a coordinated management control center to provide comprehensive, high-quality services to end users. Due to remote deployment, cloud computing cannot complete real-time data processing and analysis, thus cloud computing cannot meet the QoS demands of real-time tasks.

In the structure of fog computing, Fog nodes have average computational and storage capabilities, suitable for initial processing tasks or real-time applications. They play the role of processing data close to the source (at the network edge) to reduce latency and offload the cloud layer. Therefore, the problem of scheduling fog nodes plays a very important role, and it is necessary to focus on the allocation and management of computational tasks on fog nodes in the most efficient manner.

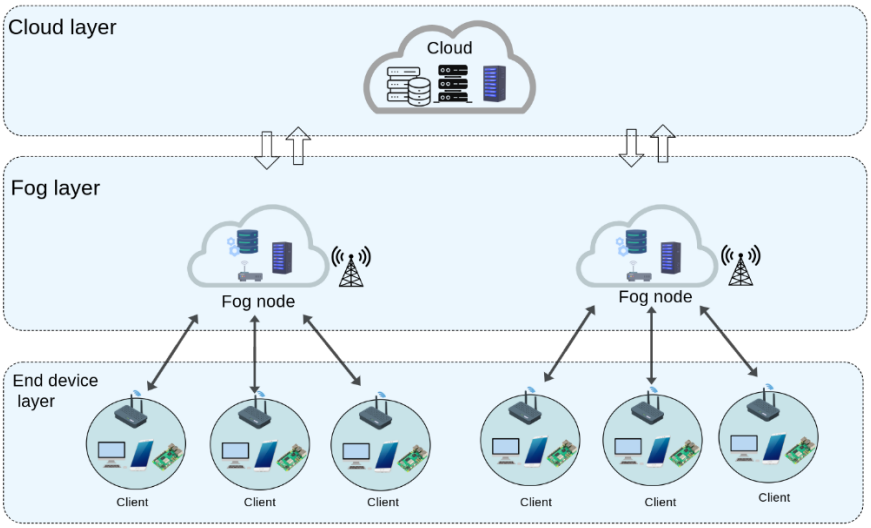

Figure 1. Hierarchical model of fog computing architecture

3.2 Memetic alrorithm

Memetic (MA: Memetic Algorithm) (Afsar, 2022) is an optimization algorithm combining the evolutionary method and local search to solve difficult and complex problems [10]. MA has the ability to overcome some limitations of traditional evolutionary algorithms and utilizes the "knowledge" of previous evolutionary generations to guide the search process for good individuals in subsequent generations. In the MA, each individual is a solution to the problem, satisfying the constraints posed by the problem. To create an individual, each evolutionary generation performs the steps: selection, crossover, and mutation. The computational steps of MA are like GA; however, the improvement of MA lies in the addition of the local search method to the computational steps. The local search method (Local Search) has the role of refining individuals after expanding the search space to individuals neighboring the current individual. Therefore, this method can find and generate better individuals; not only that, but Local Search also helps the algorithm converge faster and improves the quality of the solution compared to the traditional GA algorithm.

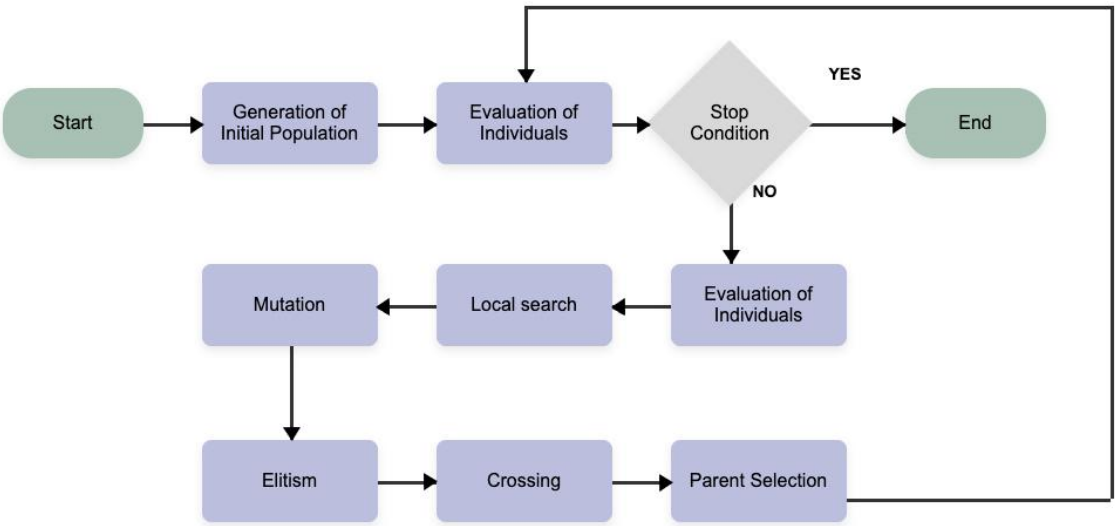

Figure 2. Hybrid Memetic algorithm

The Memetic Algorithm performs the following steps:

Step 1. Generation of Initial Population: Initially, create a population with a specified number of individuals. Each individual is a task allocation scheme (makespan).

Step 2. Evaluation of Individuals: Calculate the value of each individual in the population based on the objective function posed by the problem.

Step 3. Stop condition: If the stop condition is satisfied, the algorithm terminates and returns the best solution. If not, the algorithm proceeds to the subsequent steps.

Step 4. Parent Selection: Select suitable individuals for crossover to create new individuals in the next generation.

Step 5. Crossover: Combine individuals from the previous generation to create new individuals. This process uses information from multiple individuals participating in the crossover to generate a new individual, with the new individual inheriting the characteristics of the individuals that produced it.

Step 6. Elitism: Apply the elitism strategy to preserve the number of best individuals from the current generation to the next generation.

Step 7. Mutation: Add random attributes to the new individual after crossover.

Step 8. Local Search: Perform computations with neighboring individuals to search for better individuals.

Step 9. Recording (Acceptance): Record the individual of the current evolutionary generation if it is better than the previous generation.

Repeat the steps from (2) to (9) over multiple generations until the stop condition is satisfied.

3.3 Local Search

As mentioned in the previous section, the advantage of the memetic algorithm over other evolutionary algorithms lies in its integration of local search techniques to achieve superior performance in solving complex combinatorial optimization problems. In [11], a standard local search algorithm was presented, based on a local search problem. This algorithm has been widely recognized and utilized in numerous studies, such as [12], and [13]. Specifically, the standard local search algorithm has the following pseudo-code:

Standard Local Search

```
1.  Begin
2.      Generate an initial solution s for the problem x;
3.      Repeat Until (a local optimum is reached)
4.      Do
5.          Use s and x to generate the next neighbor n_{x,s};
6.          If (n_{x,s} is better than s) Then
7.              s := n_{x,s};
8.          EndIf
9.      EndDo
10.     Return s;
11. End
```

However, in a special environment like fog computing, where there are resource constraints and high dynamism, the Standard Local Search algorithm is truly unsuitable because it consumes a significant amount of resources and time. To explain this, we can consider its operating principle: this algorithm continuously searches until it reaches a local optimum, requiring the exploration of the entire neighborhood space and repeating the process until no further improvement is feasible. This leads to a high consumption of computational resources, resulting in an inability to meet the demands of fast and efficient response systems.

As a solution to this issue, in the paper, we present a Best Improvement Local Search algorithm. The key difference between the proposed algorithm and the Standard Local Search lies in limiting the exploration of the solution space by generating only a fixed number of random neighbors and selecting the neighbor with the best fitness value. If the best neighbor provides an improvement over the current solution, it is accepted. Conversely, the algorithm retains the initial solution and terminates. The proposed algorithm not only significantly reduces computational time and resource usage but also maintains the necessary accuracy, thereby improving the efficiency of task scheduling in the fog computing environment.

The pseudo-code of the proposed Best Improvement Local Search technique is as follows:

Best Improvement Local Search

Input: Current solution s, list of fog nodes, list of tasks, maximum number of neighbors
Output: Improved solution or s if no improvement is found

```
1.  Begin
2.      Set current_fitness = fitness_function(s, fog_nodes, tasks)
3.      Set best_neighbor = s, best_fitness = current_fitness
3.      For i from 1 to max_neighbors do
4.          Create neighbor by randomly reassigning one task in s
5.          Set neighbor_fitness = fitness_function(neighbor, fog_nodes, tasks)
6.          If neighbor_fitness < best_fitness then
7.              Set best_fitness = neighbor_fitness, best_neighbor = neighbor
8.          End If
9.      End For
10.     Return best_neighbor if best_fitness < current_fitness, else s
11. End
```

3.4 Task restructuring mechanism

In a dynamic environment like Fog computing, there always exist constraints on resources for performing tasks. These constraints can be network conditions, the computational power of Fog nodes, latency, or initial service quality requirements. Within the framework of this paper, we use the resource redistribution method for two main purposes:

- *First*, to provide a more optimal schedule if the task restructuring mechanism, yields better results. In the opposite case, retain the result that MA previously found.
- *Second,* in the case where Fog nodes are overloaded, lose connection and cannot process all tasks, or cannot participate in the processing process. The redistribution mechanism will play the role of redistributing tasks to resources (Fog nodes) in the most optimal way.

After each generation, the MA algorithm will find the best schedule (C_{best}). Based on this schedule, changes are made to the resources performing each task while still ensuring the initial resource constraints are met. This method helps expand the search space, further optimize the schedule, and most importantly, avoid local optima.

The redistribution method is performed through the following steps:

Step 1: Search for the most heavily used resource C_{best}, call L_{best}. This is the resource that finishes its work last among the resources used to execute the project.

Step 2: Iterate through each task performed by the resource L_{best} in the order of task priority, denoted as W_{best}^{R}.

Step 3: For each task found $W_i \in W_{best}^{R}$, perform the following two steps:

- Find all resources capable of performing task W_i other than the resource currently performing it, L_{best}, denoted L_W;

- Iterate through each resource capable of performing W_i, for each resource $L_{best}^{W} \in L_W$, continue with two sub-steps: (1) assign the performing resource as W_j^W, (2) Remove W_i from the list of tasks that L_{best} is currently performing

After the above step, a new schedule is obitained, and the makespan of the new schedule, $C_{newbest}$, is calculated. If it is better C_{best}, replace the makespan with $C_{newbest}$ and stop; if not, start executing the verification loop again.

$$C_{best} = \begin{cases} C_{newbest} & if\ f(C_{newbest}) < f(C_{best}) \\ C_{best} & if\ f(C_{newbest}) > f(C_{best}) \end{cases} \quad (1)$$

This computational process helps the algorithm expand the search space by creating new individuals $C_{newbest}$ based on the best individual C_{best}. As a result, when applying the restructuring method after the MA's outcome, we can obtain better individuals. Moreover, in a dynamic environment like Fog computing and IoT devices, the issue of load imbalance in task scheduling management is also addressed thanks to this restructuring approach. Due to the highly dynamic nature, the structure of Fog nodes is unstable, so scenarios such as Fog nodes losing connection, exiting, or joining anew should be taken into account. In the paper, the task restructuring mechanism, will play a role in redistributing tasks to Fog nodes without needing to rerun the algorithms from scratch. Tasks on overloaded Fog nodes will be transferred to other nodes with more suitable resources.

To support the restructuring mechanism, we integrate an algorithm for detecting overloaded or problematic nodes (the algorithm is introduced in the pseudocode section). This algorithm operates independently and continuously monitors the status of Fog nodes by checking resources (CPU, RAM, bandwidth) and responsiveness. When an overloaded node (exceeding resource thresholds) or an unavailable node (disconnected) is detected, the algorithm triggers the restructuring mechanism to adjust the task schedule. This approach ensures that the system can quickly respond to changes in the environment, maintaining performance and reliability.

Below is an example of how our proposed method works. This section will help explain the working principle of the algorithm and the restructuring method, thereby applying the proposed approach to experimental environments that are closer to real-world scenarios.

Example 1*:* In a fog computing system, 10 real tasks are allocated across 3 Fog nodes with the goal of optimizing completion time (makespan).

- Set of tasks $W = \{W_1, W_2, W_3, W_4, W_5, W_6, W_7, W_8, W_9, W_{10}\}$.
- Set of resources $L = \{L_1, L_2, L_3\}$.

The execution time is described in detail in the table and is calculated with a conversion of 1 time unit.

Table 1. Task execution time

Task	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}
Time	4	3	5	2	6	8	2	8	9	3

Fig.3 represents the allocation of tasks to Fog nodes for execution over time.

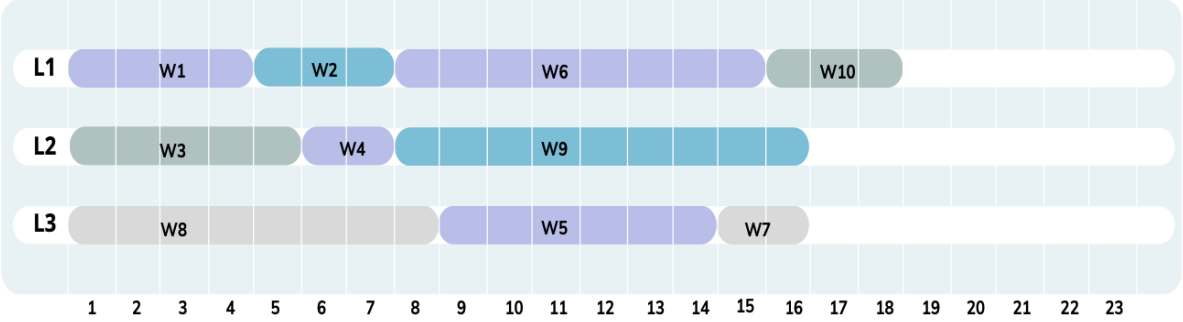

Figure 3. Diagram of task allocation to Fog nodes

With the scheme represented in Figure 3, we can represent this schedule in the form of a vector as follows:

Table 2. Task allocation to Fog nodes

Task	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}
Resource	1	1	2	2	3	1	3	3	2	1

Thus, resource L_1 will perform tasks W_1, W_2, W_5, W_{10}; resource L_2 will perform tasks W_3, W_4, W_9; resource L_3 will perform tasks W_5, W_7, W_8. With 3 Fog node and the constraints, the makespan found is 18 (time units).

Suppose the scheme described in Fig.3 is the optimal C_{best} returned by the MA algorithm, the process of restructuring C_{best} is performed to find an optimal solution (or in the case where a Fog node is overloaded, this process will play the role of balancing the task load for the Fog nodes).

The result is described in Fig.4, in which the new solution $C_{newbest}$ has a more optimal makespan compared to that of C_{best}.

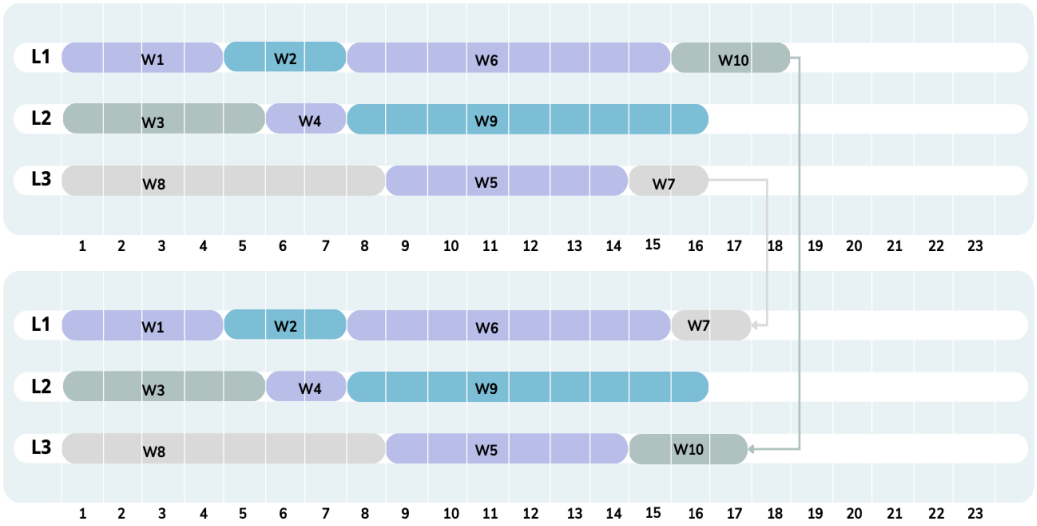

Figure 4. Result after applying schedule restructuring

In our example, a concept mentioned is the time unit, which is used as a basic measure to evaluate the execution time of tasks on fog nodes, thereby optimizing the overall completion time (makespan). This approach is quite suitable for scheduling problems focused on time optimization, especially in real-time applications. However, within the scope of task scheduling in a fog computing environment, it is not limited to just optimizing time but can also be defined based on various other criteria. These criteria depend not only on the objectives but also on the specific characteristics of each designed system. To avoid being confined within the framework of the time unit concept and to enhance the applicability of the proposed method, we will expand the concept of time into other equivalent concepts.

Considering the resource-based scheduling problem [14], we can take into account the resource unit to reflect the resource consumption level of tasks. Meanwhile, for the energy-based scheduling problem [15], the 'energy unit' can be considered as a criterion for task allocation. Or, in the priority-based scheduling problem, the criterion under consideration is the priority level, reflecting the importance of the task. In cost-based scheduling [16], the criterion examined is the cost of completing a task. Therefore, we would like to emphasize the practical applicability of this research, which is not limited to addressing scheduling problems based on time optimization but also extends to scheduling problems based on other criteria.The schedule redistribution method and the MA algorithm are described in the form of pseudocode as follows:

Algorithm 1. Detect Overloaded or Problem Nodes

Input: List of fog nodes, CPU threshold, RAM threshold, bandwidth threshold, ping timeout
Output: Lists of overloaded nodes and problem nodes

```
1. Begin
2.      Initialize overloaded_nodes, problem_nodes as empty lists
3.      For each node in fog_nodes do
5.          If node.cpu_usage > cpu_threshold or node.ram_usage > ram_threshold or
node.bandwidth_usage > bandwidth_threshold then
6.              Add node to overloaded_nodes
7.          End If
8.          If not node.is_responsive or node.processing_time > ping_timeout then
9.              Add node to problem_nodes
10.         End If
11.     End For
12.     Return overloaded_nodes, problem_nodes
13. End
```

Algorithm 2 Task restructuring mechanism,

Input: Current solution, list of fog nodes, list of tasks, resource thresholds
Output: Updated solution after task reassignment

```
1. Begin
2.      Get overloaded_nodes, problem_nodes from Detect_Overloaded_Or_Problem_Nodes
3.      For each node in overloaded_nodes + problem_nodes do
4.          For each task in node.current_tasks do
5.              Find best_node in available_nodes with min compute_completion_time(task)
6.              If best_node exists then
7.                  Reassign task from node to best_node in solution
8.              End If
9.          End For
10.     End For
11.     Return solution
12. End
```

Algorithm 3 Main Proposed Method

Input: Population size, number of generations, list of fog nodes, list of tasks, resource thresholds
Output: Best solution and its fitness value

```
1. Begin
2.      Initialize population with random solutions
3.      Set best_solution = individual with min fitness in population
4.      For each generation from 1 to generations do
5.          Create new_population with best_solution
6.          While new_population.size < popsize do
7.              Select two parents based on fitness
8.              Create child1, child2 from parents
9.              If random < LOCAL_SEARCH_RATE then
10.                 child1 = Best_Improvement_Local_Search(child1, fog_nodes, tasks)
11.             End If
12.             If random < LOCAL_SEARCH_RATE then
13.                 child2 = Best_Improvement_Local_Search(child2, fog_nodes, tasks)
14.             End If
15.             Add child1, child2 to new_population
16.         End While
17.     Update population = new_population
18.     Update best_solution if better fitness found
19.     End For
20.     Set best_solution = Reassign_Tasks(best_solution, fog_nodes, tasks, thresholds)
21.     Return best_solution, fitness(best_solution)
22. End
```

4. SIMULATION AND RESULT EVALUATION

Within the scope of this paper, we have established a simulation for the fog computing environment [17, 18] to evaluate the effectiveness of the proposed algorithm in allocating tasks from IoT devices to fog nodes. The purpose of this simulation is to assess the efficiency of the proposed algorithm. Based on this, we compared the performance of four different task allocation methods. First, we simulated a method based on simple scheduling, such as Round Robin. This is a cyclic resource allocation method that ensures each node is processed fairly within the cycle. Next, the Greedy algorithm was considered, which is an algorithm that selects the locally optimal solution at each step with the hope of achieving a globally optimal outcome. For comparison with more complex algorithms,

we sequentially examined the Genetic Algorithm (GA) and the Ant Colony Optimization (ACO) algorithm, which are algorithms with greater complexity. The simulation results section will provide a more comprehensive overview of the performance of the proposed algorithm.

Table 3. Simulation setup parameters

Parameter	Basic	Increased tasks	Increased nodes	Node overload
Number of Fog nodes	15	15	30	15 (2 overload, 2 error)
Number of tasks	600	1200	600	600
Population size	50	50	50	50
Crossover rate	0.8	0.8	0.8	0.8
Mutation rate	0.2	0.2	0.2	0.2
Maximum generations	30	30	30	30
Number of independent runs	10	10	10	10

The model is designed based on a centralized Fog Computing system, consisting of three main components: fog nodes, tasks, and the proposed algorithm (an improved Memetic Algorithm). This system aims to address the task allocation problem in the Fog Computing environment, with the goal of optimizing processing time, reducing latency, and handling situations involving node overload or failure. The simulation was conducted across four different scenarios, reflecting real-world conditions in the Fog Computing system: a baseline scenario, a scenario with an increased number of tasks, a scenario with an increased number of fog nodes, and finally, a scenario with overloaded and faulty nodes. The parameter settings for these scenarios are introduced in Table 3.

The configuration we set up Fog nodes and tasks is described through Tables 4.

Table 4. Simulation setup parameters for fog nodes and tasks

Parameter	Description	Value
Fog node setup parameters		
CPU capacity	Processing capacity of the node	Random from 2500 to 3500 (MIPS)
Bandwidth	Data transmission speed	Random from 10 to 50 (Mbps)
Current CPU load	Initial CPU load	0 (reset before each run)
Task list	List of assigned tasks	Empty (reset before each run)
Initial RAM Load	RAM load before allocation	0 (reset before each run)
RAM Capacity	Memory capacity of the node	Random from 4000 to 8000 (MB)
Task setup parameters		
Data size	Data size of the task	Random from 1 to 1000 (MB)
Processing time	Minimum processing time	Random from 0.1ms to 10ms
Current CPU load	Task priority level	Random from 0.3 to 0.9
RAM Requirement	Required RAM capacit	Random from 1 to 200 (MB)
Network Latency	Data transmission delay	Random from 1ms to 10ms

4.1 Evaluation with Round-Robin, Greedy, and GA Methods

The purpose of this section is to evaluate the effectiveness of the proposed algorithm compared to methods with lower complexity, such as Round-Robin, Greedy, and GA. In this section, we compare the metrics BEST (best value), AVG (average value), and STD (standard deviation) across four scenarios: Basic, Increased Tasks, Increased Nodes, and Node Overload. The simulation results are presented through figures 5, with three evaluation scenarios. The collected, aggregated, and evaluated results show that the proposed method outperforms the two methods GA and Round-Robin. Specifically:

- *Regarding the BEST metric*: The proposed algorithm outperforms RR, Greedy, and GA in all scenarios, with improvements ranging from 5.57% to 14.40% compared to RR, from 1.04% to

14.37% compared to Greedy, and from 7.36% to 10.98% compared to GA. This indicates that the hybrid MA is more capable of finding an optimal schedule than the other methods, especially in complex scenarios like Increased Nodes.

- *Regarding the AVG metric*: The proposed algorithm maintains stable performance with average values lower by 7.58% to 20.27% compared to RR, 9.80% to 20.22% compared to Greedy, and 7.68% to 11.85% compared to GA. This result confirms the stability of the hybrid MA across multiple runs.
- *Regarding the STD metric*: The proposed algorithm exhibits significantly lower standard deviation, with improvements ranging from 8.88% to 53.62% compared to RR, from 13.10% to 53.56% compared to Greedy, and from 7.56% to 56.65% compared to GA. This demonstrates that the proposed algorithm is less volatile, providing higher reliability in finding optimal solutions.
- *Particularly in the Node Overload scenario*: The proposed algorithm maintains good performance, surpassing other algorithms, proving its ability to handle situations effectively when resources are limited.

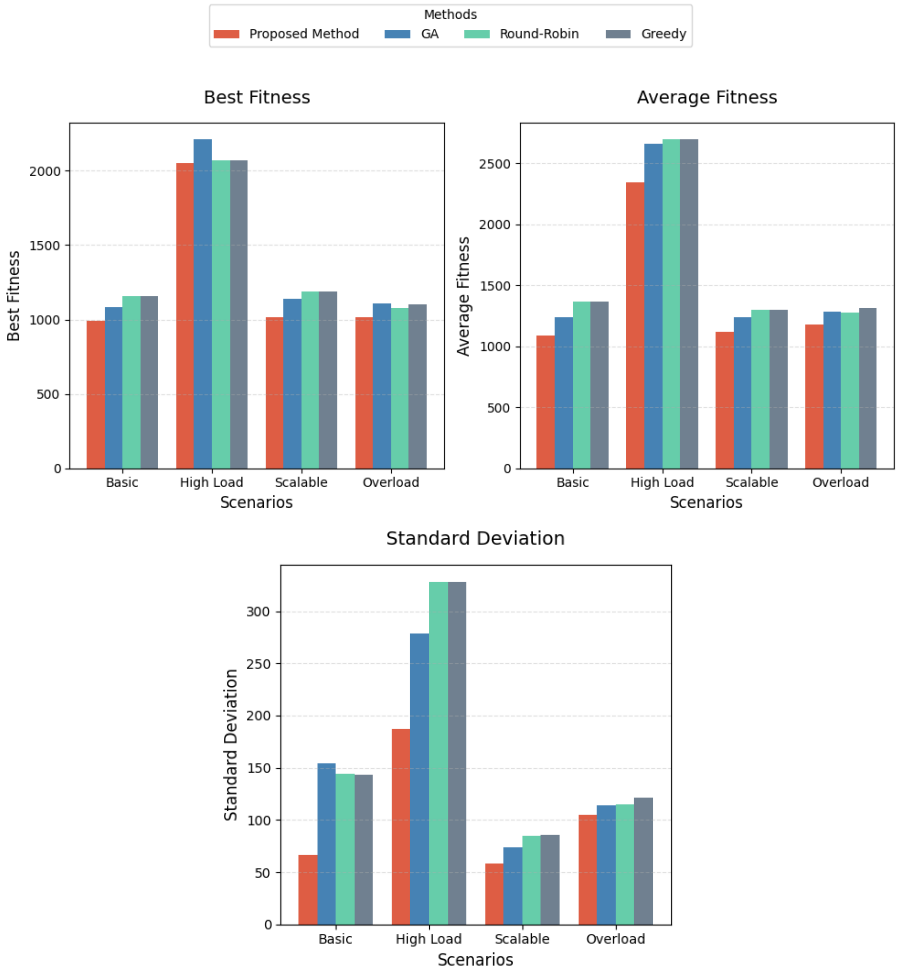

Figure 5. BEST, AVG, STD values of the proposed algorithm compared to simple algorithms GA, Round-Robin, Greedy

4.2 Comparison of Performance and Complexity with ACO

The ACO algorithm is a research method inspired by simulating the behavior of ant colonies in nature, aimed at solving complex real-world optimization problems. In this section, we will compare and evaluate the proposed algorithm with ACO based on the metrics BEST, AVG, and STD to assess performance, as well as runtime (Min Runtime, Average Runtime, Max Runtime) to evaluate complexity across the same four scenarios.

- *BEST Metric:* The hybrid MA (Proposed Algorithm) outperforms ACO in three scenarios (Basic, Increased Nodes, Node Overload) with improvements ranging from 4.17% to 12.11%. However, in the Increased Tasks scenario, ACO achieves a slightly better BEST value (-0.35%), though the difference is negligible.
- *AVG Metric:* The hybrid MA has a lower average value than ACO in all scenarios, with improvements ranging from 6.90% to 17.64%, indicating more stable performance.
- *STD Metric:* The hybrid MA exhibits a lower standard deviation than ACO in all scenarios, with improvements ranging from 12.96% to 51.58%, demonstrating superior stability.

After collecting and calculating the data, we observed that the fastest task completion time, the average completion time across all iterations, and the slowest completion time of the proposed algorithm all surpass those of ACO. Specifically:

- *Min Runtime:* The hybrid MA is faster than ACO in all scenarios, with improvements ranging from 22.11% to 52.82%.
- *Average Runtime:* The hybrid MA has a lower average runtime than ACO, with improvements ranging from 20.80% to 46.67%.
- *Max Runtime:* The hybrid MA is also faster than ACO, with improvements ranging from 14.89% to 30.90%.

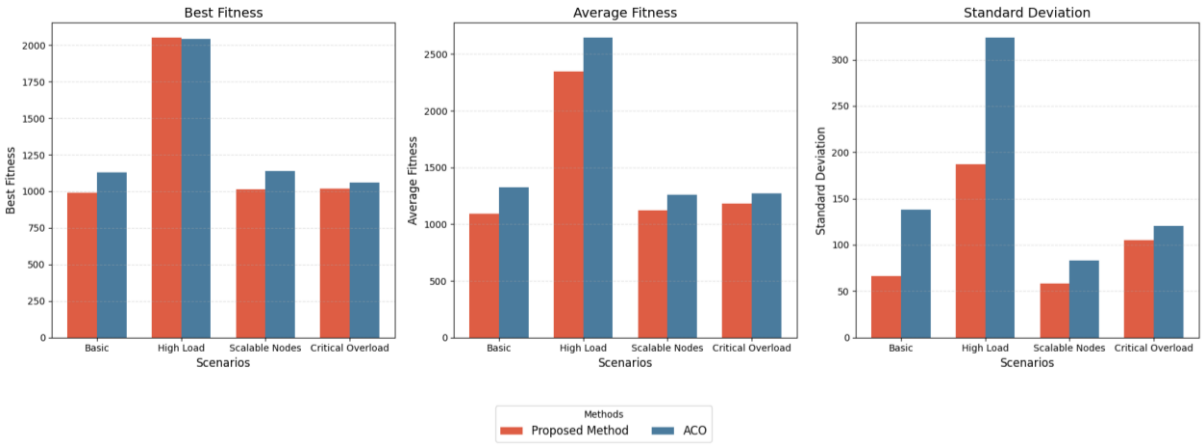

Figure 6. BEST, AVG, STD values of the proposed algorithm compared to metaheuristic algorithms ACO

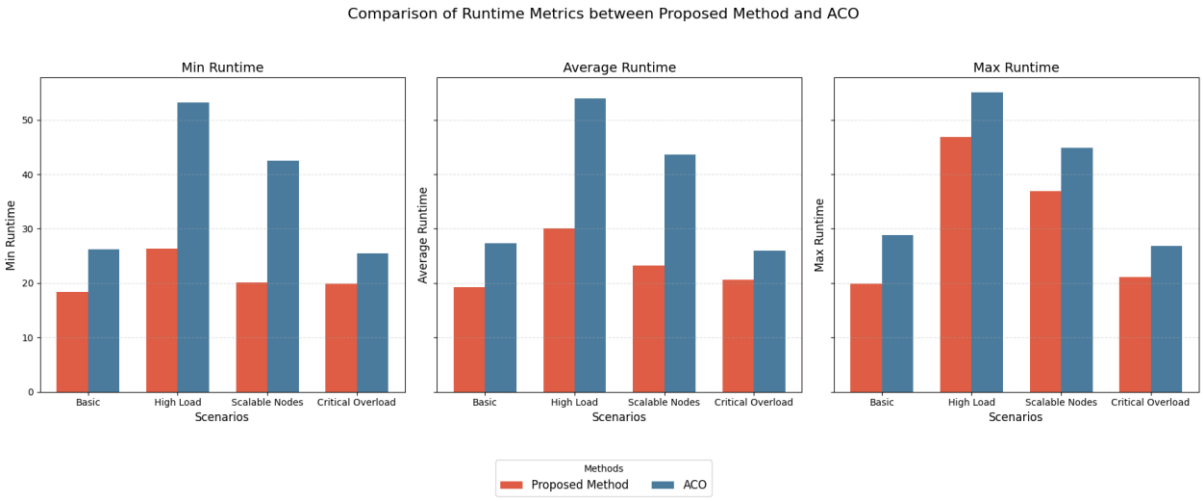

Figure 7. Minimum runtime, average runtime, and maximum runtime values of the proposed algorithm compared to the ACO

Simulation results show that the proposed algorithm outperforms the RR, Greedy, GA, and ACO algorithms in optimizing task allocation for Fog nodes. The proposed algorithm not only achieves better fitness values (lower BEST and AVG), higher stability (lower STD), but also has shorter runtime compared to ACO. Particularly in the Node Overload scenario, the proposed algorithm maintains good performance, demonstrating its ability to effectively handle resource-constrained situations. This proves that the proposed method in this paper optimizes task allocation for Fog nodes, and the restructuring approach can handle cases of overloaded Fog nodes, thereby emphasizing the feasibility of the method.

5. CONCLUSION

The proposed method in this paper combines techniques from the Memetic Algorithm and a task restructuring mechanism. Through simulation and result evaluation, it is evident that the method achieves superior performance compared to Greedy, Round-Robin, GA, and ACO in finding optimal schedules while maintaining stable performance in complex scenarios such as increased tasks or node overload. Specifically, the proposed algorithm consistently demonstrates its superiority over the two compared methods.

This improvement confirms that integrating evolutionary and heuristic methods brings significant benefits in optimizing project scheduling. This highlights the feasibility and effectiveness of combining evolutionary and heuristic approaches in optimizing task allocation, in this case, within a fog computing environment.

This study not only provides an effective solution for the scheduling problem but also opens up new development directions for hybrid algorithms in resource management and project planning. In the future, as end devices become more powerful and capable of performing deeper tasks, and as Fog nodes are equipped with greater computational and processing capabilities, integrating deep learning, reinforcement learning, or more advanced methods will yield even greater efficiency. Additionally, applying Federated Learning to the task allocation process in a fog computing environment is a promising direction.

REFERENCE

Volkov, A., Muthanna, A., Paramonov, A., Koucheryavy, A., & Elgendy, I. A. (2024). Optimized Data Transmission and Signal Processing for Telepresence Suits in Multiverse Interactions. Journal of Sensor and Actuator Networks, 13(6), 82. https://doi.org/10.3390/jsan13060082

Roman, R., Lopez, J., Mambo, M.: Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. Elsevier Future Gener. Comput. Syst. **78**(2), 680–698 (2018).

Mahmood, Z.: Fog computing concepts, frameworks and technologies. Springer Link, Book, (2018). https://doi.org/10.1007/978-3-319-94890-4

H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan and C. Maple, "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing", *IEEE Access*, vol. 7, pp. 115760-115773, 2019.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 13-16.

Shao, W.; Shao, Z.; Pi, D. A network memetic algorithm for energy and labor-aware distributed heterogeneous hybrid flow shop scheduling problem. *Swarm Evol. Comput.* **2022**, *75*, 101190.

Ren, Hualing, et al. "A memetic algorithm for cooperative complex task offloading in heterogeneous vehicular networks." *IEEE Transactions on Network Science and Engineering* 10.1 (2022): 189-204.

Fog Computing and the Internet of Things: Extend the Cloud To where the Things are, White Paper, 2016, Available online: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computingoverview.pdf (accessed on 8 April 2018).

Tang, Bo, et al. "A hierarchical distributed fog computing architecture for big data analysis in smart cities." *Proceedings of the ASE BigData & SocialInformatics 2015*. 2015. 1-6.

Afsar, S. e. (2022). Multi-objective enhanced memetic algorithm for green job shop scheduling with uncertain times. Swarm and Evolutionary Computation, 68 (2022): 101016.

D. Johnson, C. Papadimitriou, and M. Yannakakis, "How easy is local search," *Journal of Computer and System Sciences*, vol. 37, pp. 79–100, 1988.

Lin, Jih-Yiing, and Ying-Ping Chen, "Analysis on the collaboration between global search and local search in memetic computation," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 5, pp. 608-623, 2011.

Guimaraes, Frederico G., et al., "Local learning and search in memetic algorithms," *2006 IEEE International Conference on Evolutionary Computation*, IEEE, 2006.

Singh, S., & Chana, I. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14(2), 217-264.

Li, K., et al. (2018). Energy-efficient task scheduling in IoT systems: A survey. *IEEE Internet of Things Journal*, 5(5), 3871-3885.

Xu, L., et al. (2017). Cost-aware scheduling in cloud computing: A survey. *IEEE Transactions on Cloud Computing*, 5(3), 456-469.

Thang, D.V.; Volkov, A.; Muthanna, A.; Koucheryavy, A.; Ateya, A.A.; Jayakody, D.N.K. Future of Telepresence Services in the Evolving Fog Computing Environment: A Survey on Research and Use Cases. Sensors 2025, 25, 3488. https://doi.org/10.3390/s25113488

Artem, V.; Vadim, K.; Elgendy, I.A.; Muthanna, A.; Koucheryavy, A. DD-FoG: Intelligent Distributed Dynamic FoG Computing Framework. Future Internet 2022, 14, 13. https://doi.org/10.3390/fi14010013