scientific reports

Check for updates

OPEN

Dynamic multi-criteria scheduling algorithm for smart home tasks in fog-cloud IoT systems

Ruchika Bhakhar✉ & Rajender Singh Chhillar

The proliferation of Internet of Things (IoT) devices in smart homes has created a demand for efficient computational task management across complex networks. This paper introduces the Dynamic Multi-Criteria Scheduling (DMCS) algorithm, designed to enhance task scheduling in fog-cloud computing environments for smart home applications. DMCS dynamically allocates tasks based on criteria such as computational complexity, urgency, and data size, ensuring that time-sensitive tasks are processed swiftly on fog nodes while resource-intensive computations are handled by cloud data centers. The implementation of DMCS demonstrates significant improvements over conventional scheduling algorithms, reducing makespan, operational costs, and energy consumption. By effectively balancing immediate and delayed task execution, DMCS enhances system responsiveness and overall computational efficiency in smart home environments. However, DMCS also faces limitations, including computational overhead and scalability issues in larger networks. Future research will focus on integrating advanced machine learning algorithms to refine task classification, enhancing security measures, and expanding the framework's applicability to various computing environments. Ultimately, DMCS aims to provide a robust and adaptive scheduling solution capable of meeting the complex requirements of modern IoT ecosystems and improving the efficiency of smart homes.

Keywords Internet of Things, Dynamic scheduling, Multi-criteria optimization, Fog computing, Cloud computing, Smart home

Abbreviations

IoT	Internet of things
DMCS	Dynamic multi-criteria scheduling
PSO	Particle swarm optimization
GA	Genetic algorithm
ACO	Ant colony optimization
COA	Cultural algorithm
MOSA	Multi-objective simulated annealing
MoHHOTS	Multiobjective Harris Hawks optimization-based task scheduling
AEO	Artificial ecosystem-based optimization
NSGA-II	Non-dominated sorting genetic algorithm II
MGWO	Multi-objective grey wolf optimizer
EHEO	Enhanced hybrid equilibrium optimizer
SSA	Salp swarm algorithm
BWM	Best worst method
OppoCWOA	Opposing chaotic whale optimization algorithm
HEFT	Heterogeneous earliest finish time
DVFS	Dynamic voltage and frequency scaling
IWO-CA	Invasive weed optimization and cultural algorithm
QoS	Quality of service
E-AVOA-TS	Enhanced African vultures optimization algorithm for task scheduling

The rapid expansion of the IoT, particularly in smart home systems, has led to a proliferation of interconnected devices generating vast amounts of data and requiring substantial computational resources[1]. Managing these

Department of computer science and applications, Maharshi Dayanand University, Rohtak, India. ✉email: ruchikabhakhar@gmail.com

computational tasks efficiently is critical to ensure responsiveness, reliability, and user satisfaction in smart home environments.

Cloud computing has traditionally provided scalable and flexible solutions for processing and storing the immense data generated by IoT devices[2]. Its centralized resource management and scalability allow for seamless integration and management of resources across global networks. However, despite its numerous benefits, cloud computing faces significant challenges, particularly in handling latency and bandwidth constraints when processing large-scale data remotely[3]. The distance between end devices and cloud data centers can result in increased latency and network congestion, which is detrimental to time-sensitive IoT applications.

To mitigate these challenges, fog computing has emerged as a complementary paradigm that extends cloud services to the edge of the network[4]. By decentralizing and extending computational processes closer to where data originates, fog computing reduces latency and alleviates network congestion, enhancing the performance of IoT applications[5]. This approach is particularly beneficial for applications requiring real-time analytics and rapid data turnaround, such as autonomous vehicles, industrial automation, and healthcare monitoring systems[6].

In a fog-cloud computing environment, integrating both fog and cloud resources provides a balanced approach to meet the computational demands of IoT systems[7]. Fog nodes handle latency-sensitive and real-time tasks, while cloud data centers manage resource-intensive computations and long-term data storage. However, effectively scheduling tasks across these heterogeneous and distributed resources presents significant challenges due to the dynamic and unpredictable nature of IoT environments[8].

Efficient task scheduling is vital for optimizing resource allocation, minimizing response times, and maintaining a high quality of service[9]. The complexity of scheduling tasks in fog-cloud environments, characterized by varying computational needs and resource availability, calls for advanced scheduling solutions that can dynamically adapt to fluctuating conditions and optimize performance in real-time.

Metaheuristic algorithms have been extensively applied to solve the NP-hard problems associated with task scheduling in fog-cloud systems[10]. These algorithms, inspired by natural and biological processes, offer robust mechanisms for exploring complex and multi-dimensional solution spaces efficiently. Prominent among these are Genetic Algorithms (GA)[11], Particle Swarm Optimization (PSO)[12], and Ant Colony Optimization (ACO)[13], each known for specific advantages in terms of solution quality, convergence speed, and the balance between exploration and exploitation[14]. However, these algorithms often face limitations in adaptability and scalability when dealing with the dynamic conditions of fog-cloud systems[15].

To address these challenges, study propose a novel algorithm, the DMCS algorithm, which innovatively combines the strengths of PSO and ACO within a dynamic and adaptive framework tailored for the unique demands of fog-cloud computing. The DMCS algorithm is designed to dynamically switch between different optimization strategies based on continuous analysis of system states and task-specific characteristics. This flexibility ensures that the scheduling mechanism remains optimal under varying operational conditions, thereby enhancing the overall efficiency and responsiveness of IoT systems.

Further enhancing the DMCS algorithm is the integration of the Best Worst Method (BWM) for multi-criteria decision-making[16]. This method is employed to prioritize tasks based on a variety of critical factors, including urgency, resource demand, and potential impact on system performance. By incorporating BWM, the DMCS algorithm improves its decision-making capabilities, enabling more nuanced and strategic task prioritization that aligns with the operational goals of fog-cloud computing.

The key contributions of this paper are:

- A novel DMCS algorithm is developed, integrating PSO and ACO to enhance task scheduling efficiency in fog-cloud environments.
- A dynamic adaptation mechanism is introduced, which selects the most appropriate optimization strategy based on real-time system analysis.
- A multi-criteria decision-making approach using BWM is implemented to prioritize tasks, considering factors such as computational complexity, urgency, and data size.
- The performance of the proposed DMCS algorithm is evaluated through extensive simulations, demonstrating improvements in makespan, operational costs, and energy consumption compared to existing algorithms.The remainder of this paper is organized as follows: "Related work" reviews related work in task scheduling for fog and cloud computing, highlighting both their strengths and limitations. "Methodology" covers the overall methodology of the study. "System model" describes the system model and presents the mathematical formulation of the scheduling problem. "DMCS-based task scheduling" details the proposed DMCS algorithm and its components. "Experimental setup and performance evaluation" covers experimental setup and performance evaluation of the he DMCS algorithm. "DMCS algorithm validation" covers the DMCS algorithm performance validation on various benchmarking optimization test function. "Results and discussion" discusses the experimental setup and simulation results. "Discussion and limitations" covers the detailed discussion on the obtained results from the study and also covers limitations of the study. Finally, "Conclusion and future directions" concludes the study and outlines future research directions.

Related work

Task scheduling in fog-cloud computing environments is a critical challenge due to the dynamic and heterogeneous nature of IoT systems. Efficient scheduling algorithms are essential for optimizing resource utilization, reducing latency, and improving overall system performance. In this section, we review existing scheduling approaches, critically evaluating their strengths and limitations to identify gaps in the current literature.

Najafizadeh et al.[17] proposed a Multi-Objective Simulated Annealing (MOSA) algorithm to securely assign tasks to fog and cloud nodes, optimizing time and cost while controlling access levels and meeting task deadlines. Their approach effectively balances multiple objectives using a Goal Programming Approach (GPA). However, it

has limitations in scalability and flexibility when applied to larger, more complex networks, as the computational overhead increases significantly with the number of tasks and nodes.

Movahedi et al.[18] introduced an optimization framework focusing on time and energy consumption in fog computing. They proposed an Opposing Chaotic Whale Optimization Algorithm (OppoCWOA) to address task scheduling. While their method demonstrates effectiveness in reducing energy consumption and execution time, it suffers from high computational complexity due to the chaotic mapping and opposition-based learning mechanisms, which may limit its applicability in real-time systems where quick decision-making is crucial.

Yadav et al.[19] developed a hybrid approach combining the Fireworks Algorithm with the Heterogeneous Earliest Finish Time (HEFT) heuristic for task scheduling. Their bi-objective optimization aimed at minimizing makespan and cost, showing superior performance in simulations. However, the approach may have limited applicability to diverse IoT tasks due to its reliance on specific task characteristics and assumptions inherent in the HEFT heuristic, which may not hold in more dynamic or heterogeneous environments.

Hosseinioun et al.[20] proposed an energy-aware scheduling approach using Dynamic Voltage and Frequency Scaling (DVFS) in fog computing. Their hybrid algorithm, combining Invasive Weed Optimization and Cultural Algorithm (IWO-CA), effectively reduces energy consumption without violating task precedence constraints. Nevertheless, the method is limited to specific processor types that support DVFS, restricting its generalizability across different hardware platforms commonly found in heterogeneous fog environments.

Azizi et al.[21] focused on minimizing energy consumption and meeting QoS requirements by introducing a priority-aware semi-greedy algorithm with a multi-start procedure. While their model improves deadline adherence and energy efficiency, it is primarily designed for fog nodes and does not fully consider the integration with cloud resources, limiting its applicability in integrated fog-cloud environments where tasks may need to be offloaded to the cloud due to resource constraints.

Hussien[22] presented an Artificial Ecosystem-based Optimization (AEO) method enhanced by Salp Swarm Algorithm (SSA) operators for task scheduling in cloud-based IoT services. Their algorithm outperformed others in terms of makespan time and throughput. However, the scalability of the approach remains a concern for larger systems, as the complexity of the algorithm increases with the number of tasks, potentially leading to longer scheduling times.

Ghafari and Mansouri[23] addressed task scheduling challenges with an Enhanced African Vultures Optimization Algorithm for Task Scheduling (E-AVOA-TS). Their framework showed effectiveness and robustness, but the high computational cost associated with generating and evaluating a large number of candidate solutions could be a limitation in real-time applications where swift scheduling decisions are necessary.

Ali et al.[24] introduced the Multiobjective Harris Hawks Optimization-based Task Scheduling (MoHHOTS) algorithm, optimizing delay and energy consumption with significant improvements. However, its application scenarios are somewhat limited, as the algorithm may not perform optimally in highly dynamic environments due to its convergence characteristics.

Rao and Qin[25] proposed the Enhanced Hybrid Equilibrium Optimizer (EHEO) for AIoT task scheduling, showing superior performance in reducing makespan and energy consumption. The high computational cost of the algorithm, resulting from its complex equilibrium optimization process, may hinder its practicality in resource-constrained environments typical of IoT edge devices.

Mousavi et al.[26] developed a Directed Search operator in NSGA-II for IoT-based task scheduling, achieving better deadline adherence and overall performance. Despite its effectiveness, the approach may be limited to specific IoT devices and network configurations due to assumptions made in the model about the network topology and task characteristics.

Agarwal et al.[27] proposed a Hybrid Genetic Algorithm for multiprocessor task scheduling, improving efficiency. However, the high complexity of the algorithm and the possibility of premature convergence could be drawbacks for large-scale systems with a high number of tasks and processors.

Saif et al.[28] introduced a Multi-Objective Grey Wolf Optimizer (MGWO) algorithm to address cloud-fog computing challenges, significantly reducing delay and energy consumption. While effective, the high complexity of the algorithm and its sensitivity to parameter settings may limit its scalability and robustness in varying operational conditions.

Iftikhar et al.[29] presented HunterPlus, an AI-based task scheduling approach focusing on energy efficiency and job completion rate. Although promising, the method may result in high energy consumption in certain scenarios due to the overhead associated with AI computations, and it may require significant computational resources not always available in fog environments.

Recent advances in task scheduling

Task scheduling in fog-cloud computing environments is a critical challenge due to the dynamic and heterogeneous nature of IoT systems. Efficient scheduling algorithms are essential for optimizing resource utilization, reducing latency, and improving overall system performance. In this section, we review existing scheduling approaches, critically evaluating their strengths and limitations to identify gaps in the current literature.

Ghobaei-Arani et al.[30] provided a comprehensive review of resource management approaches in fog computing, including task scheduling mechanisms that aim to minimize execution time and energy consumption. While their findings highlight the effectiveness of deep reinforcement learning, they also note that the reliance on such computationally heavy methods may not be feasible for resource-constrained fog nodes.

Liu et al.[31] introduced an energy-efficient task allocation algorithm for mobile edge computing systems. Their approach addresses the heterogeneity of resources and seeks to reduce energy consumption. However, static parameter settings in their model can result in challenges in highly dynamic environments.

Sun et al.[32] developed a multi-objective optimization approach for resource scheduling in fog computing using an improved Non-dominated Sorting Genetic Algorithm II (NSGA-II). Their method successfully

optimizes makespan and energy consumption. However, the complexity of NSGA-II may hinder scalability when applied to large-scale systems, as the computational overhead increases.

Xia et al.[33] surveyed federated learning approaches for task scheduling in edge-cloud systems. Their research highlights the potential of using federated learning to enhance data privacy and reduce latency. Despite its promise, the method involves significant communication overhead and synchronization, which may not be practical in all fog-cloud environments, particularly those with unstable or limited network capacity.

Finally, Chen et al.[34] proposed a dynamic task offloading strategy for mobile edge computing with hybrid energy supply. This method effectively reduces latency and energy consumption. However, similar to previous approaches, it struggles with multi-objective optimization involving cost and resource utilization, particularly in systems with high levels of heterogeneity.

The above review underscores that while significant advancements have been made in task scheduling for fog-cloud environments, challenges related to computational complexity, scalability, and dynamic adaptability remain.

Gaps and limitations in existing research

Despite the advancements made by these studies, several gaps remain unaddressed:

- **Scalability**: Many algorithms exhibit high computational complexity, making them less suitable for large-scale or real-time systems where quick and efficient scheduling is critical.
- **Flexibility and Generalizability**: Several approaches are tailored to specific types of tasks, devices, or network configurations, limiting their applicability across the diverse and heterogeneous environments typical of IoT systems.
- **Comprehensive Multi-Objective Optimization**: Existing methods often focus on optimizing a limited set of objectives, failing to adequately balance multiple conflicting criteria such as execution time, cost, energy consumption, and quality of service (QoS).
- **Dynamic Adaptation**: There is a lack of algorithms capable of dynamically adapting to changing system conditions and task requirements in real-time, which is essential in highly dynamic fog-cloud environments.
- **Integration of Fog and Cloud Resources**: Many studies focus predominantly on either fog or cloud resources, without effectively leveraging the synergistic potential of integrated fog-cloud architectures.

Contribution

To address these unaddressed areas, study propose an Enhanced DMCS algorithm that integrates the strengths of PSO and ACO within a dynamic and adaptive framework. Our approach offers the following contributions:

- **Improved Scalability**: By employing efficient optimization strategies and reducing computational overhead, the algorithm enhances scalability, making it suitable for large-scale and real-time applications.
- **Flexibility and Generalizability**: The DMCS algorithm is designed to be applicable to a wide range of IoT tasks and devices, accommodating the heterogeneity of fog-cloud environments.
- **Comprehensive Multi-Objective Optimization**: Our approach effectively balances multiple objectives, including execution time, cost, energy consumption, and QoS requirements, through a multi-criteria decision-making process.
- **Dynamic Adaptation Mechanism**: The algorithm incorporates dynamic adaptation to adjust optimization strategies based on real-time system analysis and task characteristics, ensuring optimal performance under varying conditions.
- **Integrated Fog-Cloud Resource Utilization**: We leverage both fog and cloud resources synergistically, optimizing task allocation to maximize the benefits of the integrated architecture.By filling these gaps, the study aims to contribute a robust, efficient, and adaptable scheduling solution that enhances the performance and reliability of smart home IoT systems in fog-cloud computing environments.

Summary

The reviewed literature underscores the ongoing efforts to develop effective task scheduling algorithms for fog-cloud computing. While significant progress has been made, challenges remain in achieving scalability, flexibility, comprehensive multi-objective optimization, dynamic adaptation, and integrated resource utilization. Our proposed DMCS algorithm addresses these challenges, offering a novel solution that advances the state-of-the-art in task scheduling for fog-cloud IoT systems.

The table summarizing related papers (see Table 1) provides an overview of various scheduling algorithms proposed in recent literature, highlighting their scheduling factors, approaches, and limitations. This detailed comparison underscores the diversity of strategies employed to enhance task scheduling in fog-cloud computing environments, demonstrating the strengths and limitations of each method.

Methodology

This section details the foundation of fog-cloud computing, task scheduling, and the proposed DMCS algorithm.

Smart home architecture

In the rapidly evolving landscape of IoT, smart home applications are proliferating, driven by advances in connectivity and automation technologies. Smart homes integrate a multitude of IoT devices, such as sensors, actuators, and advanced home management systems, enabling automated control over various home functions like lighting, temperature, security, and entertainment. These devices collect vast amounts of data, which require substantial storage and processing capabilities [1].

References	Scheduling factors	Approach	Advantages	Limitations
Najafizadeh et al.[17]	Time, Cost, Access Levels, Deadlines	Multi-Objective Simulated Annealing (MOSA)	Balances multiple objectives using GPA	Limited scalability and flexibility
Movahedi et al.[18]	Time, Energy Consumption	Opposing Chaotic Whale Optimization Algorithm (OppoCWOA)	Reduces energy consumption and execution time	High computational complexity
Yadav et al.[19]	Makespan, Cost, Throughput	Hybrid: Fireworks Algorithm and HEFT heuristic	Superior performance in simulations	Limited applicability to diverse IoT tasks
Hosseinioun et al.[20]	Energy Consumption, Precedence Constraints	Hybrid Evolutionary Algorithm: IWO and Cultural Algorithm (IWO-CA)	Reduces energy without violating constraints	Limited to specific processor types
Azizi et al.[21]	Energy Consumption, Deadline Violation Time	Priority-Aware Semi-Greedy Algorithm with Multi-Start	Improves deadline adherence and efficiency	Limited to fog nodes
Hussien[22]	Makespan Time, Throughput	Artificial Ecosystem-based Optimization (AEO) enhanced by SSA	Outperforms others in makespan and throughput	Scalability concerns for larger systems
Ghafari and Mansouri[23]	Makespan, Cost, Energy Consumption	Enhanced African Vultures Optimization Algorithm for Task Scheduling (E-AVOA-TS)	Effective and robust task scheduling	High computational cost
Ali et al.[24]	Delay, Energy Consumption	Multiobjective Harris Hawks Optimization-based Task Scheduling (MoHHOTS)	Improves delay and energy consumption	Limited application scenarios
Rao and Qin[25]	Makespan, Energy Consumption	Enhanced Hybrid Equilibrium Optimizer (EHEO)	Reduces makespan and energy use	High computational cost
Mousavi et al.[26]	Deadline Satisfaction, Energy Consumption	Directed Search operator in NSGA-II	Better deadline adherence	Limited to specific IoT devices
Agarwal et al.[27]	Makespan, Energy Consumption	Hybrid Genetic Algorithm	Improved scheduling efficiency	High complexity, possible premature convergence
Saif et al.[28]	Delay, Energy Consumption	Multi-Objective Grey Wolf Optimizer (MGWO)	Reduces delay and energy consumption	High complexity, sensitive to parameters
Iftikhar et al.[29]	Energy Efficiency, Job Completion Rate	HunterPlus	Focuses on energy efficiency	High energy consumption due to AI overhead

Table 1. Summary of related papers.

Traditionally, cloud computing has been employed to manage the storage and processing needs of IoT devices, enabling end devices to offload heavy computations and data storage to cloud data centers [2]. However, as the number of devices increases and the distance to centralized cloud servers remains significant, challenges such as high latency, limited bandwidth, and network congestion become more pronounced [3]. These challenges can degrade performance and reliability, particularly in applications where real-time processing is crucial.

To address these issues, fog computing has been introduced as an intermediary layer that bridges the cloud and end devices [4]. Fog computing architectures reduce delays and enhance performance by situating computing resources closer to the source of data generation-the IoT devices themselves [5]. By decentralizing processing tasks through fog nodes, fog computing facilitates quicker response times and more efficient data handling.

As depicted in Fig. 1, the smart home architecture consists of three layers: the IoT edge devices, the fog layer, and the cloud layer. The IoT devices, such as thermostats, security cameras, and lighting systems, are connected to nearby fog nodes via local networks (e.g., Wi-Fi, Zigbee). These fog nodes act as localized processing units, handling immediate data processing tasks and serving as intermediaries between the edge devices and the cloud servers [7].

The fog layer does not merely function as a proxy server but plays a crucial role in data preprocessing, analysis, and decision-making for latency-sensitive applications. It aggregates data from multiple edge devices, performs computations, and can execute control commands back to the devices in real-time. This reduces the need to transmit all raw data to the cloud, thereby minimizing network bandwidth usage and latency [35].

The cloud layer remains pivotal for long-term data storage, intensive data analytics, and management of applications that do not require immediate processing. Fog nodes communicate with cloud servers over the internet to upload processed data, receive updates, and synchronize system-wide information [36].

The interaction between the layers is illustrated in the sequence diagram in Fig. 2. This diagram demonstrates the workflow within the system, highlighting the communication steps between the edge devices, fog nodes, and cloud servers.

In the sequence diagram (Fig. 2), the operational flow is as follows:

1. **Data Generation**: Edge devices generate data based on user interactions or environmental conditions.
2. **Data Transmission to Fog Nodes**: The edge devices send data to the connected fog nodes through local communication protocols.
3. **Local Processing at Fog Nodes**: Fog nodes process the data, perform real-time analytics, and may execute immediate control actions by sending commands back to the edge devices.
4. **Selective Data Transmission to Cloud**: Processed or aggregated data that requires long-term storage or further analysis is transmitted from fog nodes to the cloud servers.
5. **Cloud Processing and Storage**: The cloud layer performs intensive computations, data mining, and stores data for historical analysis.
6. **System Updates and Synchronization**: The cloud can send updates, configurations, or learning models back to the fog nodes, which may further disseminate necessary information to the edge devices.This hierarchical approach ensures that time-sensitive tasks are handled promptly by the fog layer, enhancing system

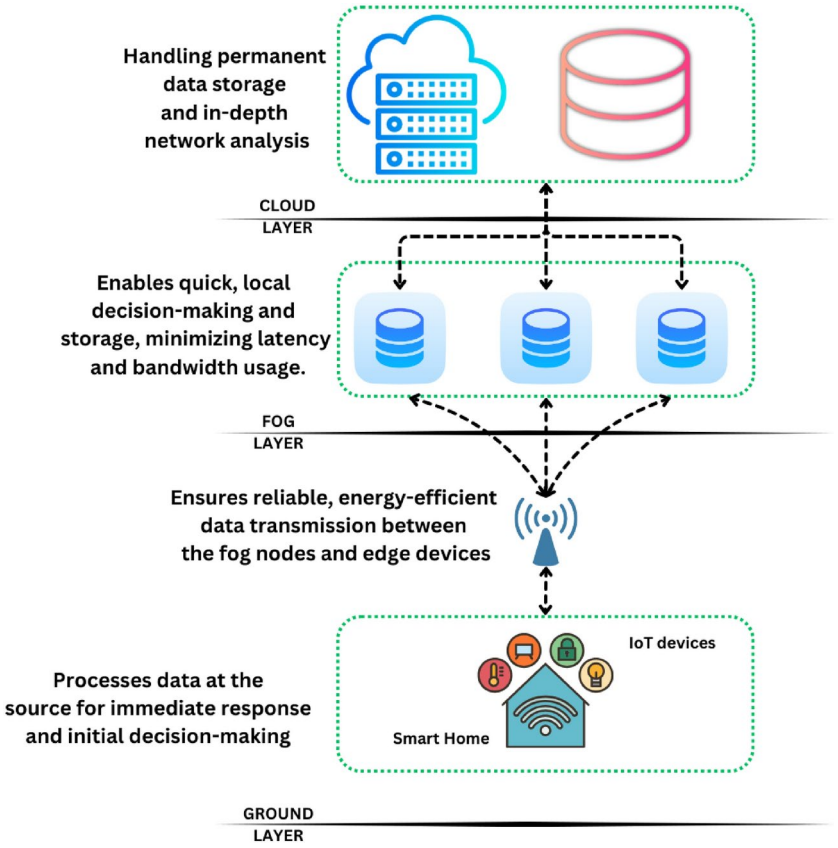

Figure 1. Architecture of a smart home integrating IoT, fog, and cloud computing.

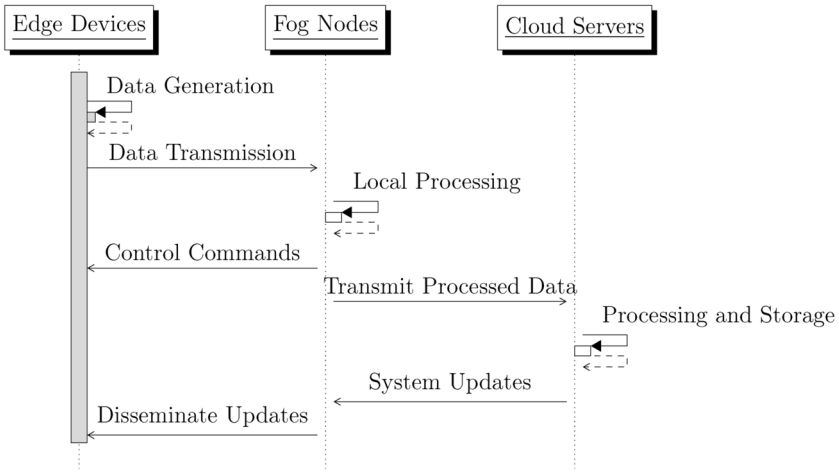

Figure 2. Sequence diagram illustrating interaction steps in the smart home architecture.

responsiveness, while the cloud layer manages tasks that require substantial computational power or are less time-critical [37].

The connection between the edge devices and fog nodes is facilitated through local area networks, enabling high-speed communication with minimal latency. Fog nodes are equipped with sufficient computational resources to handle local processing demands and are strategically placed to optimize coverage and performance within the smart home environment [38].

By effectively delineating the roles and interactions of each layer, the architecture ensures efficient utilization of resources, improved scalability, and enhanced user experience in smart home systems. The inclusion of the

sequence diagram provides a clearer understanding of the operational dynamics, allowing readers to comprehend how data flows and processing occurs within the system.

Task scheduling and resource management

Task scheduling within fog computing is crucial for aligning computational tasks with the appropriate resources to ensure efficiency and meet real-time processing demands. A fog computing task scheduler is primarily tasked with determining the optimal node for each task, ensuring that deadlines are met and service to users is seamless. This involves a sophisticated decision-making process where the fog manager node intelligently determines the best scheduling strategies, as illustrated in Fig. 3.

As depicted in Fig. 3, the task scheduling process in fog computing starts with an evaluator that assigns priorities to tasks based on incoming requests from IoT sensors. Once priorities are established, the scheduler selects an appropriate node-either fog or cloud-based on various parameters like computational power, network latency, and resource availability. Unlike traditional systems where real-time priority tasks were maintained in queues, modern fog computing environments enable dynamic scheduling directly on resource-efficient fog nodes. This shift enhances responsiveness and optimizes resource utilization across the network.

A resource manager plays a pivotal role in this process by assisting the task scheduler in determining the availability of resource-efficient nodes. This integration between task scheduling and resource management ensures that tasks are not only assigned to the most appropriate nodes but are also balanced in a way that maximizes the efficiency of the entire fog layer.

Task scheduling in fog computing is recognized as an NP-hard problem, a classification that underscores its computational complexity and the difficulty of finding optimal solutions. To address this challenge, meta-heuristic methods are employed. These methods are designed to find suboptimal but highly effective solutions to NP-hard problems. Recent advancements in this field have popularized the use of various meta-heuristic algorithms, which are favored for their simplicity and effectiveness. Notable among these are the ACO, Whale Optimization Algorithm (WOA), PSO, and the more recent African Vulture Optimization Algorithm. Each of these algorithms offers unique strengths in exploring and exploiting the search space, which is crucial for task scheduling in heterogeneous and dynamic environments like fog computing.

Dynamic multi-criteria scheduling algorithm

The DMCS algorithm is a novel meta-heuristic algorithm inspired by the complex decision-making processes found in natural systems. It is specifically designed to address the challenges of task scheduling and resource management in fog computing environments, where multiple criteria such as latency, energy consumption, and resource utilization must be simultaneously optimized. Based on the principles of multi-criteria decision-making, the DMCS algorithm dynamically adjusts its strategy according to the current state of the network and the specific requirements of each task.

The algorithm initiates with a classification phase where tasks are categorized based on their urgency and resource demands. This categorization helps in applying differentiated strategies that are tailored to the specific

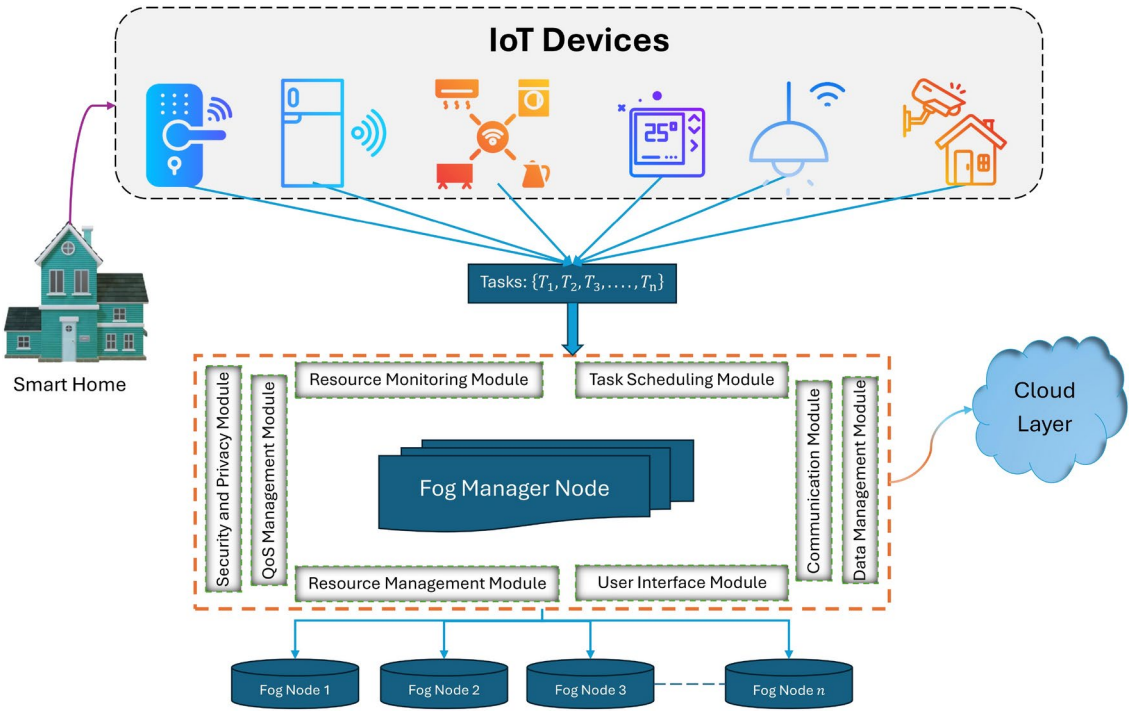

Figure 3. Task scheduling process in fog computing.

needs of each task type. Figure 4 illustrates the flow of operations in the DMCS algorithm, highlighting the dynamic interaction between different system components.

The first step in the DMCS process involves evaluating the current state of the network and the resource availability at each fog node, as shown in Fig. 4. Based on this evaluation, the algorithm assigns initial priorities to tasks using a set of predefined rules that consider both the urgency of the tasks and the current load on the network.

The task scheduler then dynamically selects the most suitable fog node for each task based on a multi-criteria scoring system. This system weighs factors such as the proximity of the node to the data source, the computational capacity of the node, and the current workload of the node. The objective is to minimize delays and balance the load across the network, thereby optimizing the overall performance and responsiveness of the system.

As tasks are executed and the network conditions change, the DMCS algorithm continuously re-evaluates and adjusts the task-node assignments. This dynamic re-scheduling capability is crucial for adapting to fluctuating workloads and changing network conditions, ensuring that performance bottlenecks are avoided and resource utilization is maximized.

In addition to task scheduling, the DMCS algorithm also incorporates a resource management component that monitors the usage of resources across the fog nodes. This component ensures that no single node becomes overburdened, which could lead to performance degradation. The resource manager works in conjunction with the scheduler to dynamically allocate and reallocate resources as needed, based on real-time data about network conditions and task performance.

The flexibility and adaptability of the DMCS algorithm make it particularly well-suited for environments with highly variable demands and heterogeneous resources, such as fog computing networks supporting IoT applications. By integrating advanced meta-heuristic techniques, the algorithm efficiently handles the complexity of multi-criteria decision-making in real-time, providing a robust solution for dynamic task scheduling and resource management in distributed computing environments.

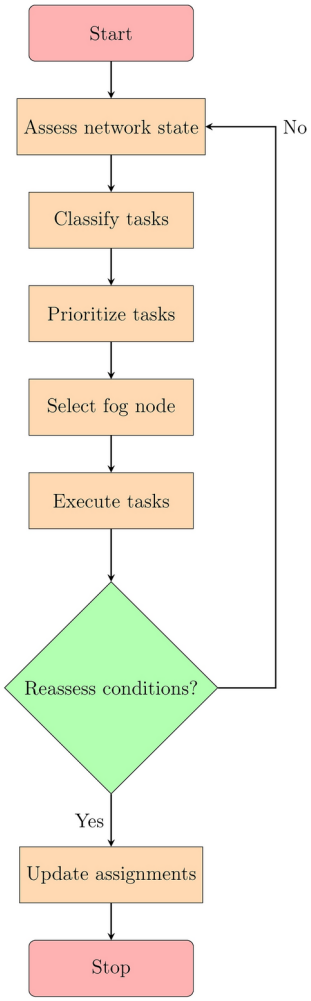

Figure 4. Flowchart illustrating the operational process of the DMCS algorithm in fog computing environments.

Particle swarm optimization (PSO)
PSO is an evolutionary computation technique developed by Kennedy and Eberhart in 1995, inspired by social behaviors observed in flocks of birds and schools of fish. The algorithm is particularly well-suited for solving continuous optimization problems and is characterized by its simplicity and effectiveness. PSO optimizes a problem iteratively by improving candidate solutions concerning a given measure of quality, typically involving minimizing or maximizing a cost function.

Each particle in the swarm represents a potential solution to the optimization problem. The particles adjust their positions by following the best-performing particles in the swarm, leading to an optimal or near-optimal solution over successive iterations.

The PSO algorithm operates on the principle of social interaction among particles, where each particle adjusts its trajectory towards its own best known position and the global best position discovered by the swarm. This behavior is inspired by the social dynamics observed in flocks of birds or schools of fish.

The specific steps of the PSO algorithm are detailed in Algorithm 1, highlighting the initialization, evaluation, update, and termination phases of the algorithm.

Initialize a swarm of particles with random positions and velocities
while termination criteria not met **do**
  for each particle in the swarm **do**
    Evaluate the fitness of the particle
    if fitness is better than the best fitness (pbest) **then**
      Update pbest
    end if
  end for
  Update the global best (gbest) from all pbests
  for each particle in the swarm **do**
    Calculate particle velocity using:
      $v_i(t+1) \leftarrow \omega v_i(t) + c_1 r_1(pbest_i - x_i(t)) + c_2 r_2(gbest - x_i(t))$
    Update particle position using:
      $x_i(t+1) \leftarrow x_i(t) + v_i(t+1)$
  end for
end while
return gbest

Algorithm 1. Particle Swarm Optimization

Each particle *i* in the swarm has a position vector x_i and a velocity vector v_i.
where:

- ω is the inertia weight that controls the impact of the previous velocity on the current velocity.
- c_1 and c_2 are cognitive and social factors, respectively.
- r_1 and r_2 are random numbers between 0 and 1.
- pbest_i is the best position of particle *i*, and gbest is the best position found by any particle in the swarm.

Ant colony optimization (ACO)
ACO is another powerful and flexible probabilistic technique for solving computational problems that can be reduced to finding good paths through graphs. This method is inspired by the behavior of ants searching for food and how they communicate their findings via pheromone trails. ACO is highly effective for discrete optimization problems such as the traveling salesman problem and network routing.

In ACO, a set of artificial ants constructs solutions by traversing a graph, guided by pheromone trails that are updated based on the quality of the solutions found. Over time, the pheromone trails lead the ants to converge on the optimal path.

The specific steps of the ACO algorithm are detailed in Algorithm 2, highlighting the initialization, solution construction, pheromone update, and optional daemon actions.

Initialize pheromone trails and place ants at starting nodes
while termination criteria not met **do**
    for each ant **do**
        Construct a solution by moving to the next node based on:
        $p_{ij}(t) \leftarrow \frac{\tau_{ij}(t)^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_k \tau_{ik}(t)^{\alpha} \cdot \eta_{ik}^{\beta}}$
    end for
    Update pheromones on all paths used by ants:
    $\tau_{ij}(t+1) \leftarrow (1-\rho)\tau_{ij}(t) + \sum \Delta\tau_{ijk}$
    Optionally perform global update or daemon actions to enhance the pheromone trail
end while
return the best solution found

Algorithm 2. Ant Colony Optimization

In the ACO algorithm, the probability π_{ij} that an ant moves from node *i* to node *j* is given by

$$\pi_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_k \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}}, \tag{1}$$

where:

- τ_{ij} is the pheromone concentration on the edge from *i* to *j*,
- η_{ij} is the heuristic value of the edge,
- α and β are parameters that control the influence of τ_{ij} and η_{ij}, respectively.As shown in Eq. (1), the probability π_{ij} is influenced by both the pheromone concentration and the heuristic value of the edge.

Integration of PSO and ACO in DMCS

The DMCS algorithm leverages the strengths of both PSO and ACO to address the challenges in fog computing environments. The integration operates on a two-tiered approach:

1. **Global Search with PSO**: PSO is utilized for global exploration of the search space, quickly identifying promising regions where optimal solutions are likely to be found.
2. **Local Search with ACO**: ACO is employed for local exploitation within the promising regions identified by PSO, refining solutions to achieve optimal or near-optimal task scheduling.This hybrid approach effectively balances exploration and exploitation, enhancing the algorithm's ability to find high-quality solutions in complex and dynamic environments.

The detailed procedure of this integration is outlined in Algorithm 3 and visualized through a flowchart in Fig. 5, illustrating the step-by-step process and interactions between PSO and ACO components.

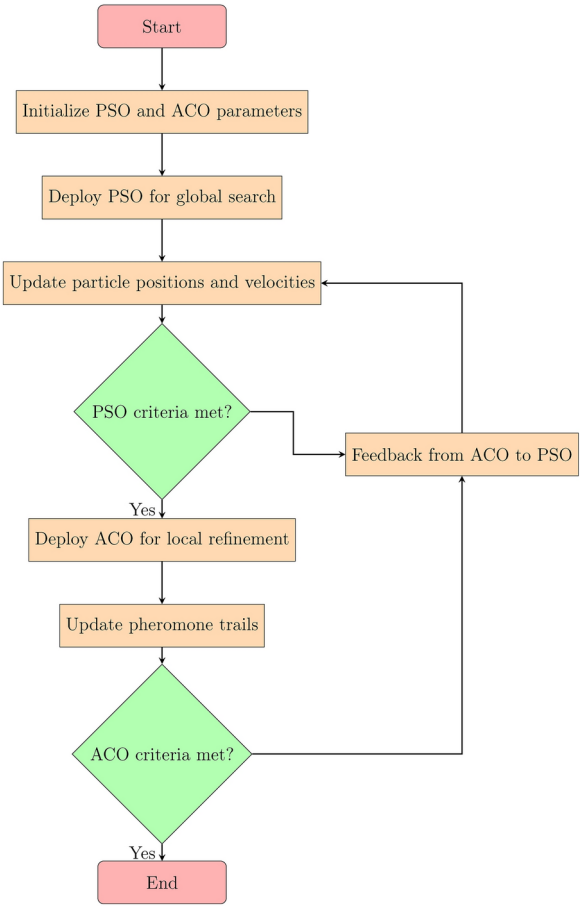

Figure 5. Flowchart illustrating the DMCS algorithm.

```
Initialize the environment and parameters
Deploy PSO for global search
Initialize particles with random positions and velocities
repeat
    for each particle do
        Evaluate the global fitness
        Update personal best and global best
        Adjust velocity and position of particles
    end for
    Deploy ACO for local search refinement
    Place ants at identified promising regions by PSO
    repeat
        for each ant do
            Construct paths based on pheromone strength
            Update pheromone trails based on quality of solution
        end for
    until local termination condition is met
    Provide feedback from ACO to PSO
until global termination condition is met
return the best solution found
```

Algorithm 3. DMCS Algorithm Integrating PSO and ACO

To ensure that the global and local searches inform each other, a feedback mechanism is implemented:

- **Feedback from ACO to PSO**: The best solutions found by ACO influence the global best (gbest) in PSO, allowing PSO to direct the swarm towards these refined solutions in subsequent iterations.
- **Solution Evaluation**: After each iteration, the combined solutions from PSO and ACO are evaluated, and the best-performing solutions are used to update the personal and global bests, influencing the subsequent behavior of both algorithms.This hybrid approach leverages the strengths of both PSO and ACO, using PSO's capability for rapid global search and ACO's proficiency in detailed local exploration and exploitation, to optimize the task scheduling and resource allocation in fog computing environments dynamically. This integration effectively addresses the complexities associated with task scheduling by balancing exploration and exploitation, thus enhancing the adaptability and efficiency of the scheduling process.

Dynamic framework for task scheduling

Algorithmic structure

In addressing the challenges of dynamic task scheduling within fog computing environments, this study introduces a robust algorithmic framework that leverages the integrated strengths of PSO and ACO. The algorithm operates through a continuous feedback loop where the global search capabilities of PSO and the local optimization strengths of ACO are synergistically combined.

The operational flow includes:

1. **Initialization**: Setting up initial parameters for PSO and ACO, and initializing the population of particles and ants.
2. **Global Optimization**: Using PSO to explore the search space and identify promising regions for task scheduling solutions.
3. **Local Refinement**: Applying ACO within the promising regions to fine-tune the task assignments and resource allocations.
4. **Feedback Mechanism**: Incorporating feedback from ACO to update PSO parameters and guide the swarm towards better solutions.
5. **Iteration**: Repeating the global and local optimization steps until convergence criteria are met.This structured approach ensures that the algorithm remains adaptable and responsive to changes in task demands and resource availability.

Adaptability and responsiveness

In the context of dynamic scheduling within fog computing environments, adaptability and responsiveness are critical characteristics that ensure the effectiveness of task allocation and resource management strategies. The DMCS algorithm enhances these capabilities through:

- **Dynamic Parameter Adjustment**: Adjusting algorithm parameters like inertia weight in PSO and pheromone evaporation rate in ACO based on real-time feedback from the system.
- **Feedback Loops**: Implementing continuous feedback between PSO and ACO to refine search strategies and improve solution quality over time.
- **Proactive and Reactive Strategies**: Combining proactive task allocation with reactive real-time adjustments to respond effectively to changing network conditions and task requirements.By integrating these mechanisms, DMCS ensures high levels of efficiency and effectiveness in dynamic and unpredictable fog computing environments.

The DMCS algorithm employs a continuous feedback loop between the global optimization carried out by PSO and the local search refinement by ACO (see Fig. 6). This feedback loop allows the system to learn and adapt from previous iterations, effectively tuning the swarm and ant behaviors to better navigate the solution space under current conditions. This not only enhances adaptability but also ensures that the system remains responsive to real-time changes.

Enhanced mutation strategy for DMCS algorithm

The introduction of an Enhanced Mutation Strategy (EMS) in the DMCS algorithm marks a significant step towards improving its optimization capability, particularly in complex multimodal landscapes. This strategy aims to increase the diversity of solutions within the population and dynamically adjusts the mutation rate based on the search's progress, allowing the algorithm to explore new and potentially more promising areas of the search space. This adaptability is crucial in complex optimization problems characterized by multiple local minima that could trap conventional algorithms.

The core idea of the EMS involves dynamically adjusting the mutation rates to enhance exploration capabilities without compromising the convergence speed. The mutation rate is modified according to a decay function, which depends on both the current iteration and the diversity of the population.

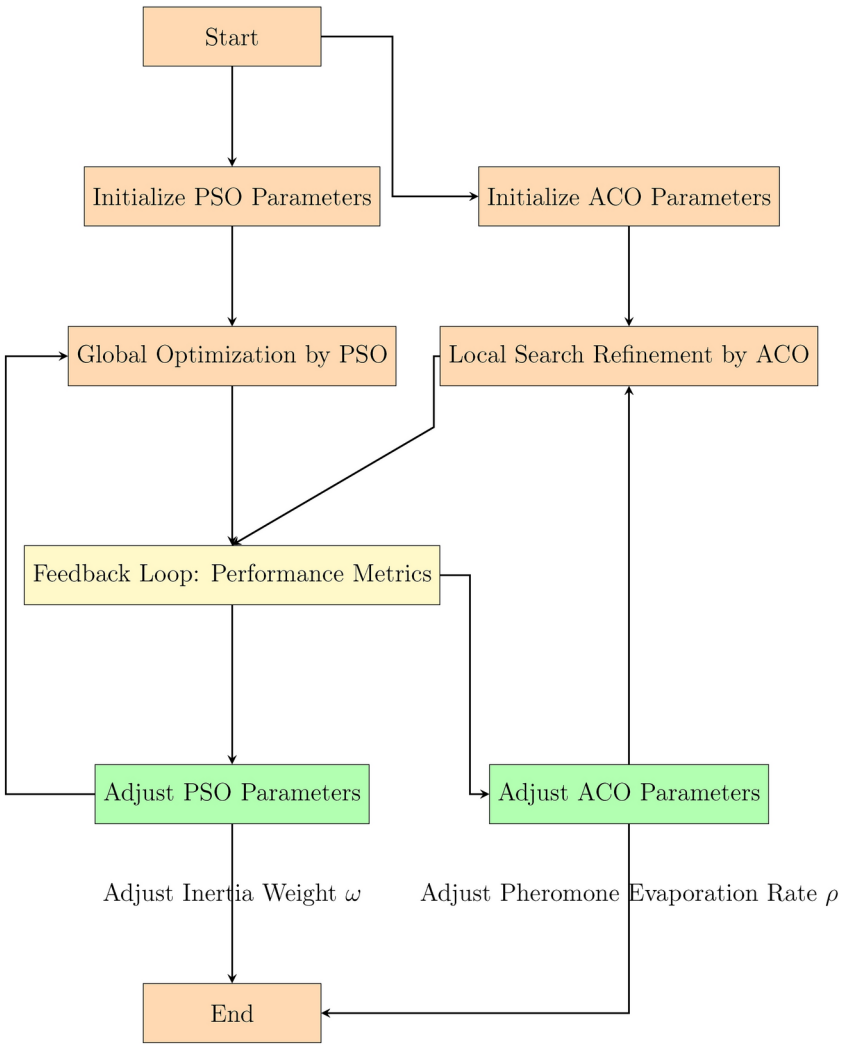

Figure 6. The continuous feedback loop between the PSO and ACO.

Algorithm 4 presents the pseudocode of the enhanced mutation strategy adopted in the study for the DMCS algorithm, and Algorithm 5 depicts how the mutation rate is dynamically adjusted against iterations.

```
Require: Population, MaxIter, MutationBaseRate
Ensure: Adapted Population
procedure ENHANCED_MUTATION(Population, iter, MaxIter)
for each individual in Population do
    if Random() < MutationRate(iter, MaxIter) then
        Mutate individual based on MutationBaseRate
    end if
end for
return Population
end procedure
```

Algorithm 4. Enhanced Mutation Strategy for DMCS

```
function MutationRate(iter, MaxIter)
return MutationBaseRate × (1 - iter/MaxIter)
end function
```

Algorithm 5. Mutation Rate Calculation

The mutation rate adjustment is given by the equation:

$$\text{MutationRate}(iter) = \text{MutationBaseRate} \times \left(1 - \frac{iter}{\text{MaxIter}}\right) \tag{2}$$

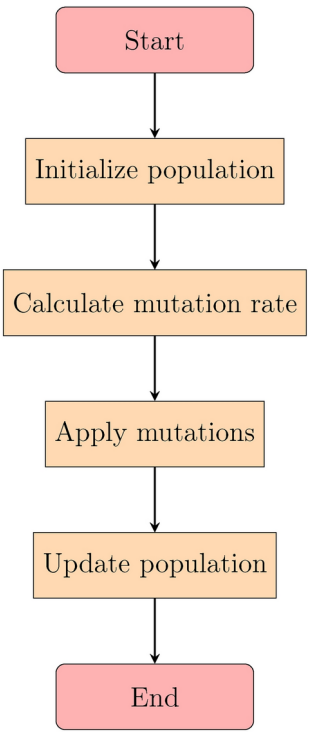

Figure 7. The enhanced mutation strategy process in the DMCS algorithm.

This equation ensures that the mutation rate decreases as the number of iterations increases, promoting higher exploration in the initial phases and more exploitation in the later stages, as shown in Eq. (2).

Empirical results from testing the DMCS algorithm with the EMS on benchmark functions such as the Sphere and Ackley functions show marked improvements. The strategy facilitated a more extensive exploration of the search space, leading to faster convergence towards the global optimum. However, functions like Rosenbrock and Rastrigin, known for their rugged landscapes, showed that while the EMS helped in escaping local minima, it also introduced stochastic noise, leading to fluctuations in the convergence trajectory. This behavior underscores the need for a balanced approach in mutation strategy to effectively navigate complex landscapes.

Figure 7 presents a flowchart that illustrates the EMS process.

The effectiveness of the EMS could be further enhanced by incorporating adaptive mechanisms that dynamically adjust the parameters based on feedback from the optimization process. Future research should focus on refining this strategy, possibly through adaptive mechanisms that respond to the state of the search, thereby optimizing the exploration-exploitation balance dynamically. Such advancements could broaden the applicability of DMCS in solving a wider array of complex optimization problems in real-world applications.

System model

In this section, we present the system model for our Enhanced MCS algorithm in fog-cloud IoT systems. The system model encompasses the task model, resource model, and the mathematical formulation of the scheduling problem. This comprehensive model lays the foundation for the DMCS-based task scheduling approach described in subsequent sections.

Task model

We consider a set of independent tasks $T = \{T_1, T_2, \ldots, T_n\}$ generated by IoT devices in a smart home environment. Each task T_i is characterized by the following attributes:

- Len_i: The length of task T_i in millions of instructions (MI).
- Mem_i: The memory requirement of task T_i in megabytes (MB).
- $\mathrm{DataSize}_i$: The size of input data for task T_i in MB.
- $\mathrm{Deadline}_i$: The deadline for task T_i to be completed.
- $\mathrm{Priority}_i$: The priority level of task T_i, determined using the BWM based on criteria such as urgency, computational demand, and data size.Tasks are heterogeneous in nature, varying in computational complexity and time sensitivity. The task model captures these variations to enable effective scheduling decisions.

Resource model

The fog-cloud computing environment comprises a set of computing nodes $M = M_{\mathrm{fog}} \cup M_{\mathrm{cloud}}$, where:

- $M_{\mathrm{fog}} = \{M_1, M_2, \ldots, M_{m_f}\}$ represents the set of fog nodes.
- $M_{\mathrm{cloud}} = \{M_{m_f+1}, M_{m_f+2}, \ldots, M_m\}$ represents the set of cloud nodes.
- $m = m_f + m_c$ is the total number of computing nodes, where m_f is the number of fog nodes and m_c is the number of cloud nodes.Each computing node M_j is characterized by the following attributes:
-
- CPU_j: The processing capacity of node M_j in MIPS (Million Instructions Per Second).
- MemCap_j: The total memory capacity of node M_j in MB.
- BW_j: The bandwidth of node M_j in Mbps.
- $C_{\mathrm{cpu},j}$: The cost per unit CPU usage on node M_j.
- $C_{\mathrm{mem},j}$: The cost per unit memory usage on node M_j.
- $C_{\mathrm{bw},j}$: The cost per unit bandwidth usage on node M_j.
- α_j: The energy consumption rate when node M_j is active.
- β_j: The energy consumption rate when node M_j is idle.
- Delay_j: The communication delay associated with node M_j.Fog nodes are located closer to the IoT devices, offering lower latency but limited computational resources. Cloud nodes provide higher computational power but are associated with higher communication delays.

Mathematical formulation

The objective of the DMCS-based task scheduling problem is to optimally assign tasks to computing nodes to minimize makespan, total cost, and energy consumption while satisfying the tasks' deadlines and resource constraints.

Decision variables

We define a binary allocation matrix Q of size $n \times m$, where:

$$Q_{ij} = \begin{cases} 1, & \text{if task } T_i \text{ is assigned to node } M_j, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

As shown in Eq. (3), the matrix Q indicates the assignment of tasks to nodes.

Constraints

1. **Task Assignment Constraint**: Each task must be assigned to exactly one computing node, as expressed in Eq. (4):

$$\sum_{j=1}^{m} Q_{ij} = 1, \quad \forall i = 1, 2, \ldots, n. \tag{4}$$

2. **Resource Capacity Constraints**: The total resource demand on each node must not exceed its capacity, as defined in Eq. (5):

$$\sum_{i=1}^{n} \mathrm{Mem}_i \times Q_{ij} \leq \mathrm{MemCap}_j, \quad \forall j = 1, 2, \ldots, m. \tag{5}$$

3. **Deadline Constraints**: The completion time of each task must not exceed its deadline, as specified in Eq. (6):

$$\mathrm{CT}_i \leq \mathrm{Deadline}_i, \quad \forall i = 1, 2, \ldots, n. \tag{6}$$

where CT_i is the completion time of task T_i.

Objective functions

We consider three objective functions:

1. Makespan Minimization

The objective is to minimize the makespan MK, as defined in Eq. (7):

$$\mathrm{MK} = \max_j \left(\sum_{i=1}^{n} \mathrm{ExeT}_{ij} \times Q_{ij} \right). \tag{7}$$

Additionally, the execution time ExeT_{ij} of task T_i on node M_j is calculated using Eq. (8):

$$\mathrm{ExeT}_{ij} = \frac{\mathrm{Len}_i}{\mathrm{CPU}_j}. \tag{8}$$

2. Total Cost Minimization

The total cost C_{tot} includes computational cost C_{comp}, communication cost C_{comm}, and deadline violation cost C_v, as expressed in Eq. (9):

$$C_{\mathrm{tot}} = C_{\mathrm{comp}} + C_{\mathrm{comm}} + C_v. \tag{9}$$

Computational Cost is defined by Eq. (10):

$$C_{\mathrm{comp}} = \sum_{i=1}^{n} \sum_{j=1}^{m} (C_{\mathrm{cpu},j} \times \mathrm{ExeT}_{ij} + C_{\mathrm{mem},j} \times \mathrm{Mem}_i) \times Q_{ij}. \tag{10}$$

Communication Cost is given by Eq. (11):

$$C_{\mathrm{comm}} = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{\mathrm{bw},j} \times \mathrm{DataSize}_i \times Q_{ij}. \tag{11}$$

Deadline Violation Cost is represented in Eq. (12):

$$C_v = \sum_{i=1}^{n} \mathrm{Penalty}_i \times \max\left(0, \frac{\mathrm{CT}_i - \mathrm{Deadline}_i}{\mathrm{Deadline}_i}\right). \tag{12}$$

3. Total Energy Consumption Minimization

The objective is to minimize the total energy consumption E_{tot}, as defined in Eq. (13):

$$E_{\text{tot}} = \sum_{j=1}^{m} \left(\text{ActiveTime}_j \times \alpha_j + (\text{MK} - \text{ActiveTime}_j) \times \beta_j \right), \tag{13}$$

where ActiveTime_j is calculated using Eq. (14):

$$\text{ActiveTime}_j = \sum_{i=1}^{n} \text{ExeT}_{ij} \times Q_{ij}. \tag{14}$$

Composite objective function
We combine the three objectives into a single composite fitness function, as shown in Eq. (15):

$$F_{\text{obj}} = \lambda_1 \times \text{MK} + \lambda_2 \times E_{\text{tot}} + \lambda_3 \times C_{\text{tot}}, \tag{15}$$

where $\lambda_1, \lambda_2, \lambda_3$ are weighting factors such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

DMCS-based task scheduling

In this section, we elaborate on the DMCS-based task scheduling method, which leverages an integrated approach using PSO and ACO to efficiently manage and allocate tasks in the fog-cloud environment. The DMCS algorithm aims to minimize the composite objective function defined in the system model by assigning tasks to the most appropriate computing nodes.

Overview of DMCS algorithm

The DMCS algorithm operates by iteratively improving a population of potential solutions (task assignments) using PSO and ACO mechanisms. It incorporates the task priorities determined by the BWM to guide the search towards optimal solutions.

To ensure effective scheduling, tasks are prioritized using the BWM. This method systematically evaluates and ranks tasks based on multiple criteria such as urgency, resource demand, and potential impact on system

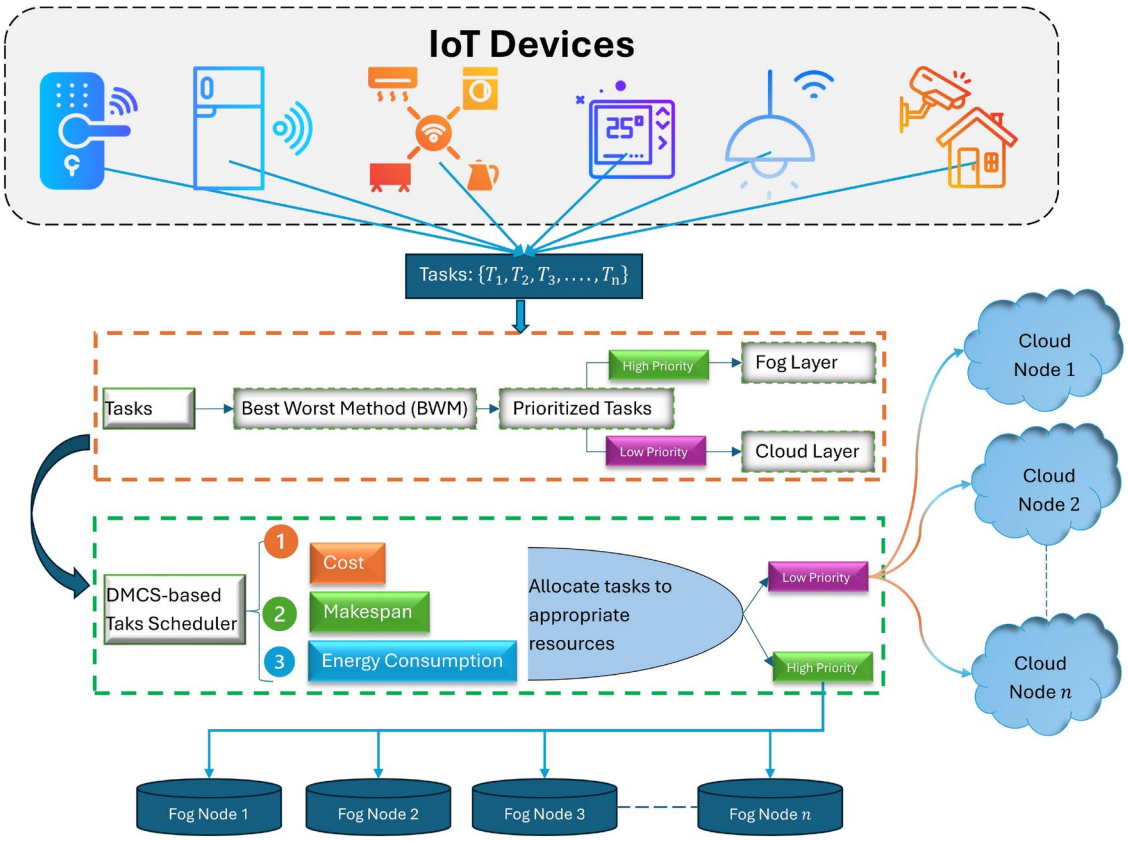

Figure 8. Task scheduling process in fog computing.

performance. By identifying high-priority and low-priority tasks, DMCS can allocate resources more effectively, directing the most critical resources to the tasks that require them most urgently.

Architecture of DMCS for task scheduling

The architecture of the DMCS for task scheduling in fog-cloud environments is designed to optimize the allocation and execution of tasks across distributed computing resources, as shown in Fig. 8. This section provides a detailed overview of the architectural model, demonstrating how DMCS prioritizes and schedules tasks efficiently through an integrated system of components and decision-making processes.

Task Manager: At the core of the DMCS architecture is the task manager, which serves as the initial point of contact for all incoming tasks. It receives a list of tasks for execution, accepting each task and analyzing its significant parameters such as count, characteristics, urgency, and resource requirements.

Smart Gateway: The smart gateway acts as an intermediary between end-devices and the network edge. It assesses tasks forwarded by the task manager and calculates their priorities based on predefined criteria. These criteria may include latency sensitivity, computational demand, and data security requirements.

Resource Manager and Task Scheduler: The resource manager and the task scheduler are pivotal in the DMCS architecture. After receiving task information and resource details from the smart gateway, these components work in tandem to allocate tasks to the most suitable nodes. The task scheduler, using information about node status and resource capacity from the resource manager, strategically dispatches tasks to fog nodes or cloud servers based on their evaluated priorities and the current load on each node.

To facilitate effective multi-criteria decision-making in task scheduling, DMCS incorporates the BWM, as described in the "System model" section. This method helps in determining the optimal node (fog or cloud) for each task based on multiple criteria, such as task size, deadline, computational intensity, and latency sensitivity. BWM ranks these criteria to ensure that latency-sensitive tasks are primarily allocated to fog nodes, which are closer to the end-users, while less time-critical tasks may be directed towards the cloud for cost-effective processing.

- *Task Submission:* Tasks received at the smart gateway are classified into latency-sensitive and latency-tolerant categories.
- *Priority Calculation:* Using BWM, tasks are prioritized. High-priority tasks, especially those requiring immediate processing, are tagged for fog nodes.
- *Resource Allocation:* The resource manager evaluates the available capacities of fog and cloud resources. Based on this evaluation, the task scheduler assigns tasks to appropriate nodes, ensuring that the system's overall load is balanced and performance criteria are met.
- *Execution:* Tasks are executed on their assigned nodes. Fog nodes handle real-time, critical tasks, maximizing responsiveness and reducing latency. Cloud nodes manage heavy-duty tasks that require significant computational resources but are less time-sensitive.
- *Result Management:* Once tasks are processed, results are relayed back to the users through the smart gateway. This gateway also acts as a broker, managing the communication between fog and cloud nodes and the end-users.Continuous feedback from the execution nodes (fog and cloud) to the task manager and resource scheduler helps in dynamically adjusting task priorities and allocations based on real-time performance data. This adaptive mechanism ensures that DMCS can respond effectively to changes in task demands or resource availability, optimizing throughput and minimizing delays and costs.

Algorithm steps

1. **Initialization**:
   - Compute the tasks' weights using BWM based on criteria such as deadline, task size, and data size.
   - Sort tasks in decreasing order of weight (priority).
   - Generate an initial population of solutions (particles/ants), where each solution represents a possible assignment of tasks to nodes.
2. **Iteration**: For each iteration $t = 1$ to $\mathrm{MaxIter}$:
   (a) **Fitness Evaluation**: For each solution in the population:
   - Calculate the fitness value using the composite objective function F_{obj}.

   (b) **Update Best Solutions**:
   - Update the global best and local best solutions based on fitness values.

   (c) **Generate New Solutions**:
   - Use PSO and ACO operators to generate new candidate solutions by exploring and exploiting the search space.

   (d) **Selection**:
   - Select the best solutions to form the population for the next iteration.

3. **Termination**:

   - The algorithm terminates after reaching the maximum number of iterations or when the solutions converge.

4. **Output**:

   - Return the best-found solution representing the optimal task assignment.

Integration with system model

The DMCS algorithm utilizes the system model to evaluate the fitness of solutions and ensure that all constraints are satisfied. By integrating the task and resource models, the algorithm can make informed decisions about task assignments, balancing the load between fog and cloud nodes while optimizing for the defined objectives.

Representation of solutions

In the DMCS framework, each potential task scheduling solution is represented by a particle in PSO and an ant path in ACO. These representations are visualized through an allocation matrix, as described in the "System model" section. Each scheduling solution involves a multi-dimensional matrix of size $n \times m$, where n is the number of tasks and m is the number of nodes.

The allocation matrix for each solution has binary values (0 or 1) as shown in Fig. 9, where:

Given the dual nature of the fog-cloud environment, DMCS effectively balances task assignments between fog and cloud nodes, ensuring that tasks are distributed optimally based on their priority and resource requirements.

Operational mechanisms

The DMCS algorithm employs both exploration and exploitation phases to efficiently search the solution space:

- In the **exploration phase**, new potential task-node mappings are explored to avoid local optima.
- The **exploitation phase** refines these mappings to optimize the fitness function.Adjustments and updates to task mappings are made dynamically based on real-time performance metrics, ensuring that the system adapts to changes in task demands and network conditions.

Task priority using BWM in DMCS

As detailed in the "System model" section, the BWM is strategically employed to prioritize tasks effectively. This integration enhances the scheduling process by efficiently ranking tasks based on multiple relevant criteria.

Integrating BWM into the DMCS framework empowers the task scheduling process with a powerful tool for making informed, efficient, and effective prioritization decisions. By methodically assessing tasks based on systematically derived and weighted criteria, DMCS ensures that resources within fog-cloud computing environments are utilized optimally, enhancing overall system performance and user satisfaction.

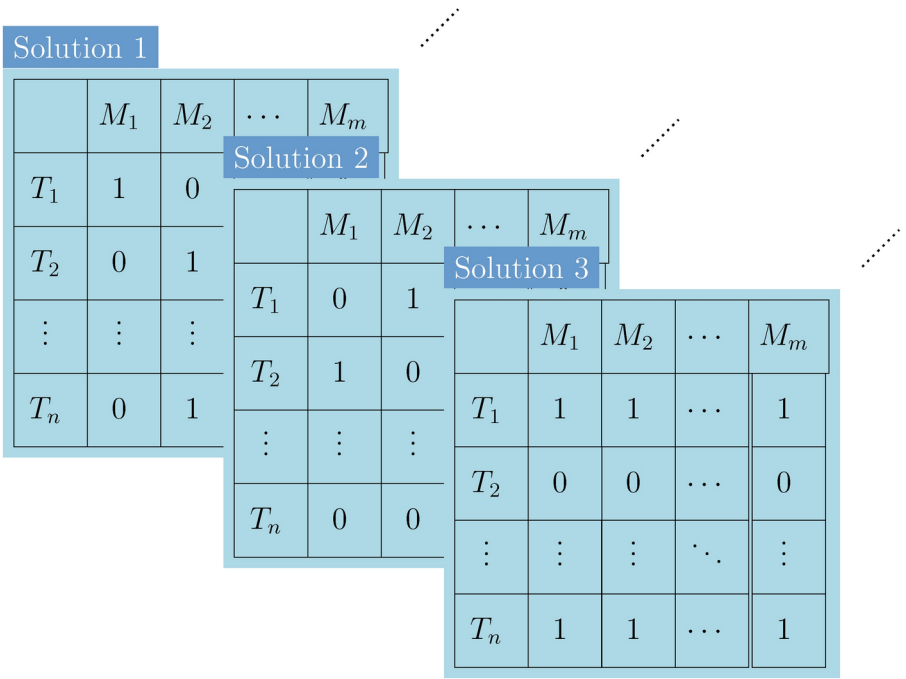

Figure 9. Allocation matrices representing different solutions.

Algorithm	Parameter	Value
PSO	Population size	100
	Max iterations	1000
	Inertia coefficient	0.9
	Cognitive coefficient	2
	Social coefficient	2
GA	Population size	100
	Max iterations	1000
	Crossover rate	0.8
	Mutation rate	0.05
ACO	Number of ants	100
	Max iterations	1000
	Decay rate	0.6
	Alpha	1
	Beta	2
COA	Number of nests	100
	Max iterations	1000
	Abandon probability (pa)	0.25
DMCS	Population size	100
	Max iterations	1000
	Initial mutation rate	0.1
	Mutation decay	Adaptive (decreasing)

Table 2. Parameter settings for algorithms.

Hardware features	Content
Processor	Intel Core i5-1335U (up to 4.6 GHz with Intel Turbo Boost Technology, 12 MB L3 cache, 10 cores, 12 threads)
Memory	16 GB DDR4-3200 MHz RAM (2 x 8 GB)
Storage	1 TB PCIe NVMe M.2 SSD
Operating System	Windows 11 Home
Software	Microsoft Office Home & Student Edition 2021
Display	68.6 cm (27") diagonal, FHD touch, IPS, three-sided micro-edge, glossy, Brightness: 300nits, Color Gamut: 72% NTSC, Resolution: 1920 x 1080
Graphics	Intel UMA Graphics
System Type	64-bit operating system, x64-based processor
Software Edition	Windows 11
Evaluation Tool	MATLAB
Version	R2024a

Table 3. The system configuration for the simulation environment.

Experimental setup and performance evaluation

Simulation environment

The simulation environment for evaluating the DMCS algorithm is established using MATLAB software. The parameters for the DMCS algorithm and other comparative algorithms are set to ensure fair and consistent comparisons across all experiments. The general settings include a population size of 100 and a maximum of 1000 iterations per run (see Table 2). The performance of each algorithm is averaged over 30 independent runs. Specific parameters for each optimizer are maintained as per their original configurations to maintain the validity of the comparison.

The software and hardware configurations for the simulation environment are presented in Table 3.

Scenario design

The DMCS algorithm is tested across a variety of task scheduling scenarios in a simulated fog-cloud computing environment. Each scenario is designed to test the algorithm under different loads and operational conditions including variations in task size, complexity, and deadline requirements. Scenarios also vary the distribution of tasks between fog and cloud layers to simulate real-world application conditions.

Metrics for performance evaluation

To assess the effectiveness of the DMCS algorithm, several metrics are considered:

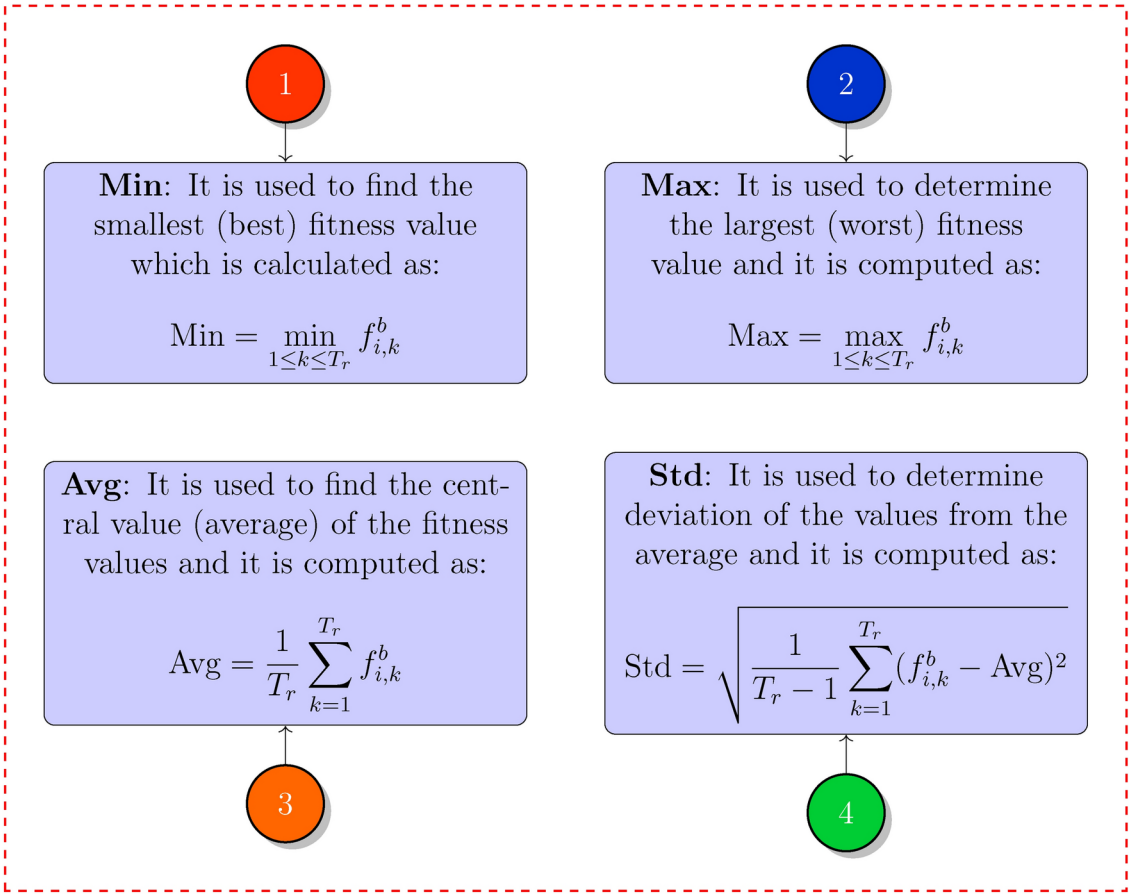

Figure 10. Illustration of performance metrics for DMCS algorithm.

- **Makespan** - The total time taken from the start of the first task to the completion of the last task.
- **Energy Consumption** - Total energy consumed by the system during the task execution.
- **Cost** - Total operational cost including computation and communication costs.
- **Deadline-Satisfied Tasks (DST%)** - The percentage of tasks that meet their deadline requirements.These metrics provide a comprehensive view of the performance and efficiency of the scheduling algorithm under various conditions as shown in Fig. 10.

Figures and additional data are presented to illustrate the performance measures, showing average, standard deviation, maximum, and minimum values of the fitness function for each experimental run.

The integration of these metrics and scenarios provides a robust framework for evaluating the performance and adaptability of the DMCS algorithm in a dynamic and distributed computing environment.

This detailed setup ensures that the DMCS algorithm is thoroughly tested and validated under conditions that mimic real-world operational scenarios in fog-cloud computing environments.

DMCS algorithm validation

The DMCS algorithm undergoes a rigorous validation process through a series of classical benchmark test optimization functions, essential for assessing its efficiency across different aspects of computational optimization. These functions are categorized into monomodal, multimodal, and composite types, each chosen to test specific capabilities of the algorithm. Monomodal functions, which possess a single global optimum, assess the algorithm's ability to exploit the search space effectively, honing in on and refining solutions around the global optimum. Multimodal functions, characterized by multiple local optima, evaluate the algorithm's exploration skills, testing its ability to navigate through complex solution spaces and avoid premature convergence to local optima. Composite functions combine features of both, offering a comprehensive challenge that evaluates the algorithm's overall efficiency in diverse optimization scenarios.

The DMCS is rigorously tested using classical benchmark functions known for their robustness in evaluating optimization algorithms. These functions simulate real-world scenarios where both the precision of the solution and the computational efficiency are crucial. Performance metrics such as the best, worst, average, and standard deviation of the results are calculated over multiple independent runs to ensure the reliability of the outcomes against the algorithm's stochastic nature.

Comparative analysis is conducted where DMCS's performance is juxtaposed with established optimization techniques like Particle Swarm Optimization, Genetic Algorithms, and others. Detailed tables and graphical representations document this analysis, highlighting how DMCS fares against these algorithms across various test functions. Special emphasis is placed on scenarios where DMCS outperforms competitors, particularly in managing complex and high-dimensional optimization challenges. These results underscore the robustness of DMCS and its adaptability for effective task scheduling in environments like fog-cloud computing, where tasks and resource demands are highly variable.

The pseudocode of the DMCS algorithm is also provided, detailing the procedures involved in the optimization process. This includes the initialization phase, parameter settings, execution of the optimization cycles, and the iterative updates of solutions based on evaluation criteria. The algorithm's capability to dynamically adapt to varying task demands and resource availabilities in a simulated environment is showcased, reinforcing its practical utility and effectiveness.

In summary, the validation section meticulously demonstrates the technical efficacy of the DMCS algorithm, establishing a solid basis for its deployment in sophisticated scheduling tasks and providing a reliable framework for future enhancements in the dynamic field of computational optimization.

Test optimization benchmark functions

The DMCS algorithm has been rigorously tested using a suite of classical benchmark optimization functions that are widely recognized for their robustness in testing various aspects of computational algorithms shown in Table 4. These functions are crucial for evaluating the algorithm's efficiency in exploring and exploiting the search space under various complexities and dimensions.

A comprehensive set of unimodal, multimodal, and fixed-dimension multimodal functions was employed to assess the algorithm's performance. The unimodal functions, which include benchmarks like Sphere, Rosenbrock, and Rastrigin, are primarily used to test the exploitation capabilities of the algorithm, focusing on its ability to hone in on the global optimum without being distracted by local optima. These functions have a single global optimum, making them ideal for evaluating the precision and speed of convergence of the DMCS.

Multimodal functions, such as Ackley, Griewank, and Levy, present multiple local optima, challenging the DMCS to demonstrate its exploration capabilities. These functions are crucial for testing the algorithm's ability to escape local optima and effectively search large and complex landscapes for the global optimum.

The fixed-dimension multimodal functions provide a controlled environment to further test the DMCS under scenarios with a known number of variables and complexities. These functions are particularly useful for benchmarking the algorithm's performance in a constrained dimensionality, assessing both exploration and exploitation strengths.

Table 4 provides an overview of the benchmark functions used, detailing their mathematical expressions, search boundaries, and objectives:

The performance of the DMCS algorithm was rigorously evaluated using a variety of benchmark functions, each with unique landscapes to challenge the algorithm's exploitation and exploration capabilities. Figures 11 and 12 depict the contour and 3D visualizations of the functions, respectively, illustrating the complexity of the landscapes the algorithm navigates during optimization. Figure 11 presenting 2D contour plot where optima can be visualized in two dimensional space. Figure 12 depicts 3D surface plot of the benchmark test functions with optima and objective function nature in the three dimensional search space.

These visualizations provide intuitive insights into the search space's complexity, where each function presents distinct challenges. For instance, the Rastrigin function features a large number of local minima, making it ideal for testing the algorithm's ability to avoid local optima traps. Similarly, the Levy function's rugged landscape tests the robustness of DMCS's exploration strategies.

Each function was tested over multiple runs, with the DMCS algorithm configurations set to optimize for the lowest values. Performance metrics such as average, best, worst, and standard deviation of the results were calculated to provide a comprehensive view of the algorithm's capabilities across these diverse functions. This

Function name	Formula	Search bound	Optimum function value	Optima location
Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	[−100,100]	0	$0 = (0, 0, \ldots, 0)$
Rosenbrock	$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	[−30,30]	0	$1 = (1, 1, \ldots, 1)$
Rastrigin	$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)]$	[−5.12,5.12]	0	$0 = (0, 0, \ldots, 0)$
Ackley	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	[−32,32]	0	$0 = (0, 0, \ldots, 0)$
Griewank	$f(x) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	[−600,600]	0	$0 = (0, 0, \ldots, 0)$
Levy	$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1}(w_i - 1)^2$ $[1 + 10\sin^2(\pi w_i + 1)] + (w_n - 1)^2[1 + \sin^2(2\pi w_n)]$	[−10,10]	0	$1 = (1, 1, \ldots, 1)$

Table 4. Benchmark test optimization functions for DMCS validation.

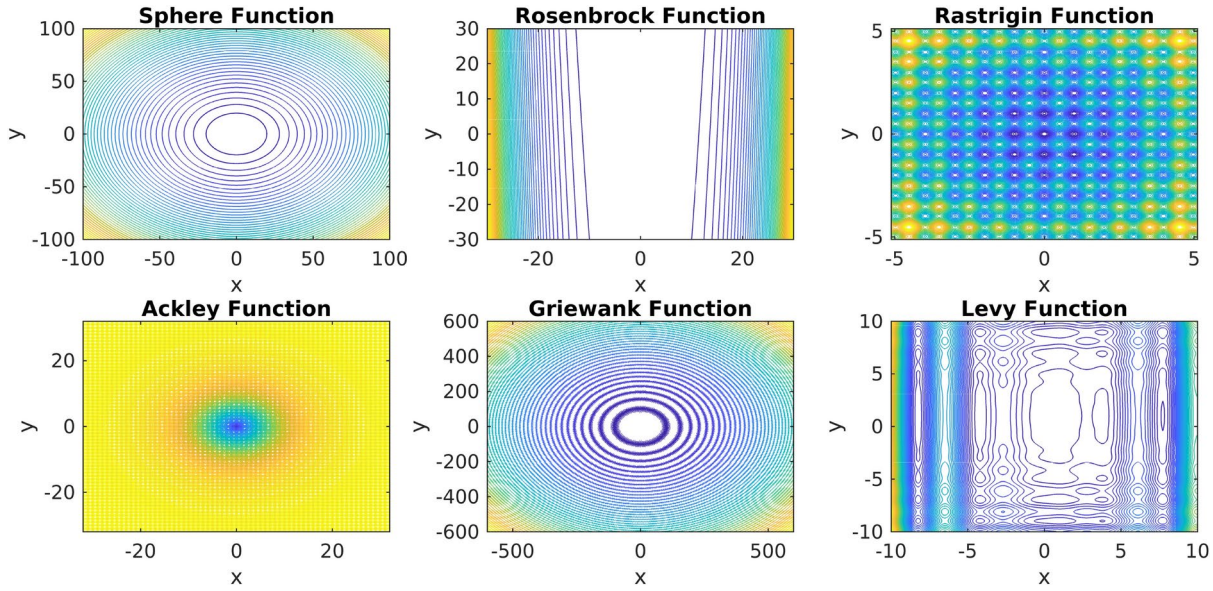

Figure 11. Contour plots of benchmark functions used in the evaluation of the DMCS algorithm. From top left to bottom right: Sphere, Rosenbrock, Rastrigin, Ackley, Griewank, and Levy functions.

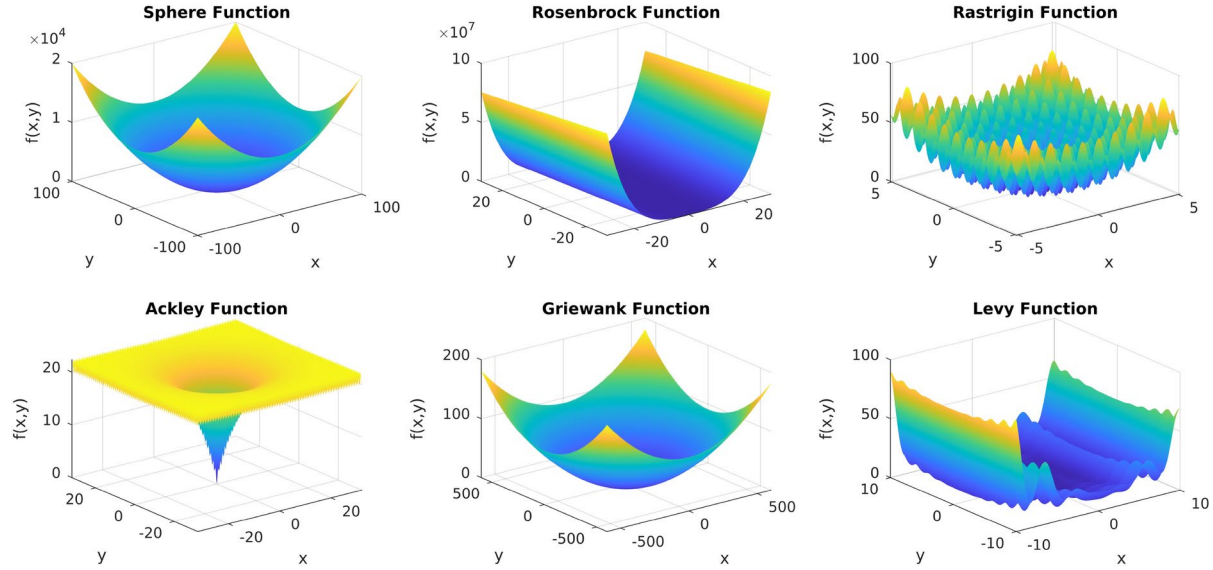

Figure 12. 3D surface plots of the benchmark functions demonstrating the varied topographies, including peaks, valleys, and plateaus.

rigorous testing ensures that the DMCS algorithm is not only theoretically sound but also practically effective across a range of complex optimization scenarios.

Results and discussion

Convergence analysis

This section evaluates the performance of the DMCS algorithm, particularly focusing on its convergence behavior across various classical benchmark functions. The enhanced mutation strategy implemented in DMCS is examined through its impact on the convergence rates, depicted in the convergence plots (see Fig. 13).

The convergence analysis demonstrates that DMCS, while integrating an enhanced mutation strategy, exhibits varied performance across different test functions. For instance, in the Sphere function, the DMCS algorithm shows rapid convergence towards the global optimum, outperforming other algorithms such as PSO and ACO in early iterations. This indicates a robust search capability in smoother landscapes, where the risk of local minima is minimal[39,40].

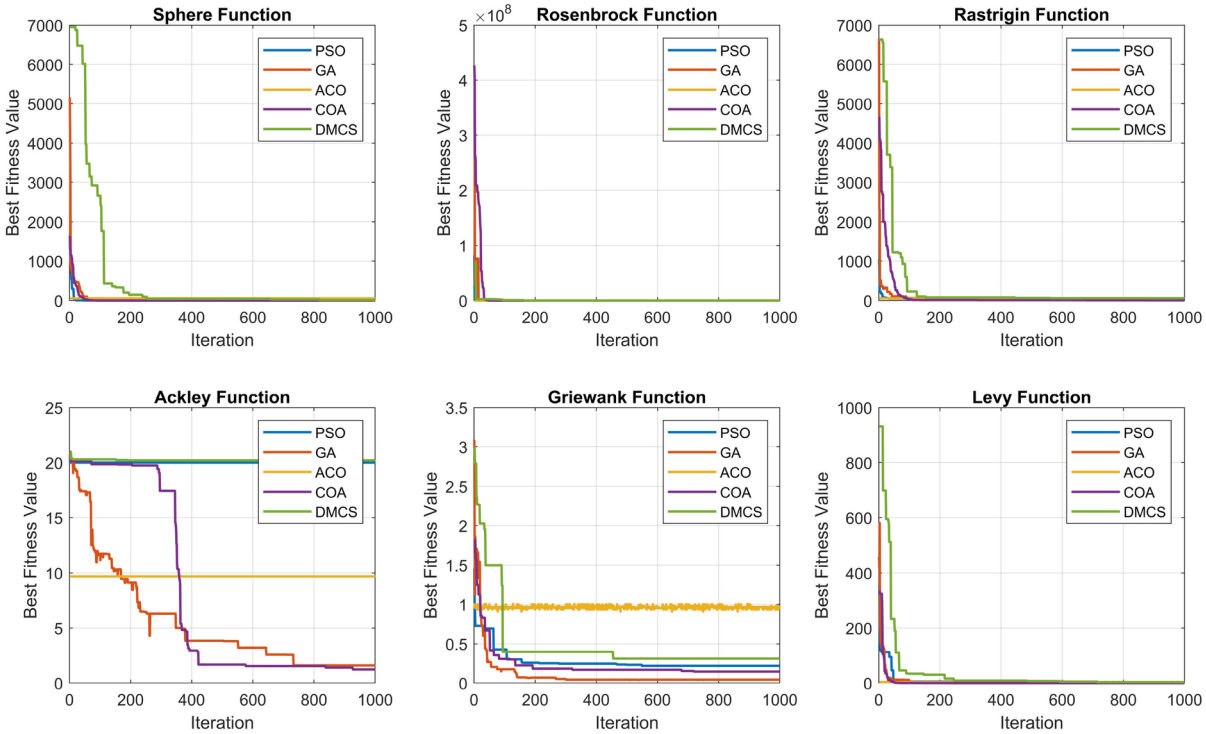

Figure 13. Convergence plots illustrating the optimization progress of DMCS compared to other conventional algorithms across different test functions. The DMCS algorithm shows varying levels of performance, with notable improvements in simpler test functions and challenges in more complex scenarios.

In contrast, the Rosenbrock and Rastrigin functions, known for their deceptive and complex landscapes, presented challenges. The DMCS algorithm initially lagged behind algorithms like GA and ACO but eventually reached competitive fitness values towards the later stages of the iterations. This behavior underscores the strength of the enhanced mutation strategy in escaping local optima and exploring the search space more thoroughly[41,42].

Particularly notable is the performance on the Griewank and Levy functions, where DMCS demonstrated significant improvements in convergence speed and stability. The algorithm effectively balanced exploration and exploitation phases, as evidenced by its steady and consistent convergence curves. This is a testament to the efficacy of the enhanced mutation strategy in managing the exploration-exploitation trade-off, which is critical in multimodal and high-dimensional search spaces[43].

However, the analysis also highlights some limitations. For example, in the Ackley function, where the landscape features numerous local minima, the DMCS algorithm showed slower convergence relative to more specialized algorithms like GA, which could navigate such complex terrains more effectively. This suggests that while the enhanced mutation strategy improves diversity and search breadth, it may require further tuning to optimize performance in highly rugged and multimodal landscapes.

Overall, the convergence plots (Fig. 10) illustrate that the DMCS algorithm, equipped with the enhanced mutation strategy, offers a competitive and reliable option for tackling a broad range of optimization problems. The results encourage further refinements and adaptations of the mutation parameters based on the problem's specific characteristics to optimize both convergence rates and solution accuracy.

In summary, the DMCS algorithm represents a promising approach in the field of evolutionary computation, with its enhanced mutation strategy proving to be particularly beneficial in diverse and challenging benchmark scenarios. Future work will focus on adaptive mechanisms to further refine this strategy, aiming to achieve optimal performance across all types of benchmark functions.

DCMS algorithm response analysis

The DMCS algorithm's response analysis through trajectories in the first dimension provides a comprehensive insight into its behavior under various test conditions. As illustrated in Fig. 14, the trajectory plots for functions like Sphere, Rosenbrock, Rastrigin, Ackley, Griewank, and Levy demonstrate distinct patterns of agents' movements across iterations. These plots serve as a qualitative metric for understanding the algorithm's exploration and exploitation dynamics in both unimodal and multimodal function landscapes[41].

The trajectory analysis of the Sphere function reveals rapid convergence towards optimal regions, indicating efficient exploitation capabilities of DMCS. Conversely, in the Rastrigin function, the trajectories spread out across the search space, reflecting the algorithm's ability to explore extensively, crucial for avoiding local minima in complex multimodal landscapes.

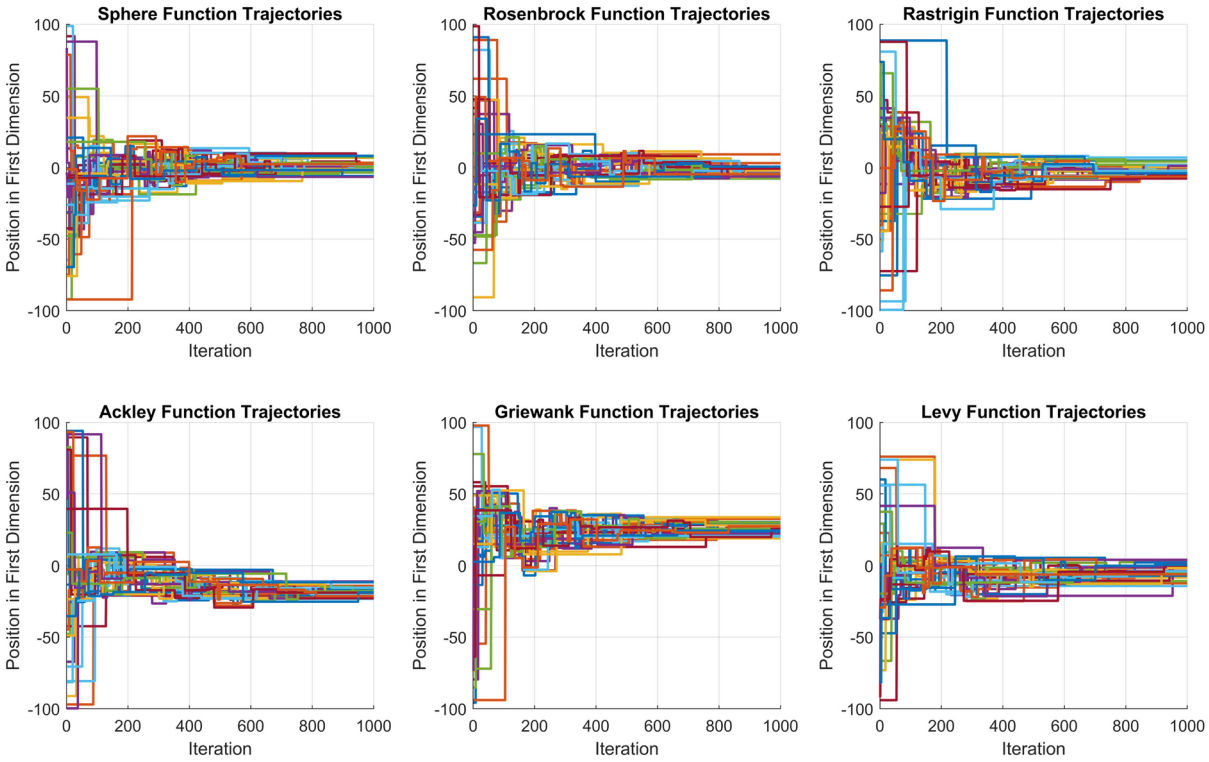

Figure 14. Trajectory analysis of DMCS algorithm showing agent movements in the first dimension across different test functions.

The Rosenbrock and Ackley function trajectories, however, exhibit periodic and somewhat erratic movements, suggesting challenges in navigating the functions' deceptive gradients and flat regions. This behavior underscores the algorithm's responsiveness to the function's topological challenges, adapting its search strategy between exploration and exploitation as needed.

Moreover, the trajectory plots for the Griewank and Levy functions highlight the algorithm's sustained exploration efforts, with occasional convergence spikes indicating successful local optima identification followed by diversification. This pattern is crucial for problems where the search space contains numerous local optima, and premature convergence could lead to suboptimal solutions[43].

In conclusion, the DMCS algorithm demonstrates a balanced approach to handling diverse function landscapes, adapting its mutation and crossover strategies dynamically to optimize both exploration and exploitation. Future iterations of this algorithm could focus on enhancing adaptive mechanisms to further refine this balance, potentially incorporating machine learning techniques to predict and react to the complexities of the search landscape dynamically. This ongoing adaptation is expected to improve the robustness and efficiency of the DMCS algorithm, making it a strong candidate for solving complex optimization problems across various real-world applications.

Comparison of standard DMCS and enhanced mutated DMCS

The evaluation of the DMCS algorithm and its enhanced version employing a mutation strategy (Enhanced Mutated DMCS) showcases distinct performance traits across various benchmark functions. This comparative analysis, illustrated through convergence plots (see Figures 15 and 16), highlights the efficacy and adaptability of the mutation strategy in handling complex optimization landscapes.

The DMCS algorithm, depicted in Fig. 15, demonstrates consistent performance across functions like Sphere and Ackley, where it swiftly converges to the global optimum. This rapid convergence indicates a strong exploitation capability within relatively simple landscapes. However, for more complex functions like Rosenbrock and Rastrigin, DMCS shows delayed convergence, suggesting challenges in navigating rugged multimodal terrains[44,45].

In contrast, the Enhanced Mutated DMCS, shown in Fig. 16, reveals an improved trajectory in these complex functions. The mutation strategy introduces variability in the population, thereby preventing premature convergence and encouraging thorough exploration of the search space. This is particularly evident in the Rosenbrock and Rastrigin functions, where Enhanced Mutated DMCS not only converges faster but also maintains a stable trajectory towards the optimum[46,47].

These observations are substantiated by the trajectory plots from various functions (see Fig. 17), where the first dimension's movement across iterations is plotted for both versions of the algorithm. The trajectory plots for Enhanced Mutated DMCS display more dynamic and varied paths in the early iterations, which gradually

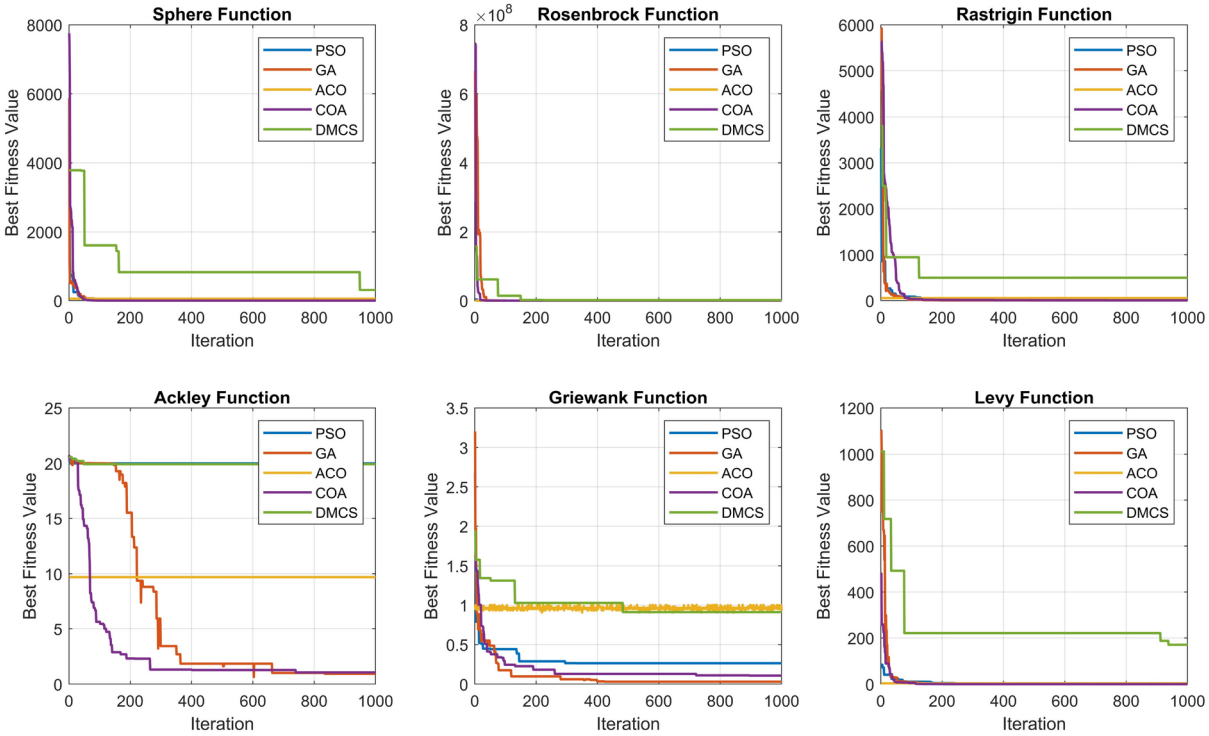

Figure 15. Convergence plots for the standard DMCS algorithm across various benchmark functions.

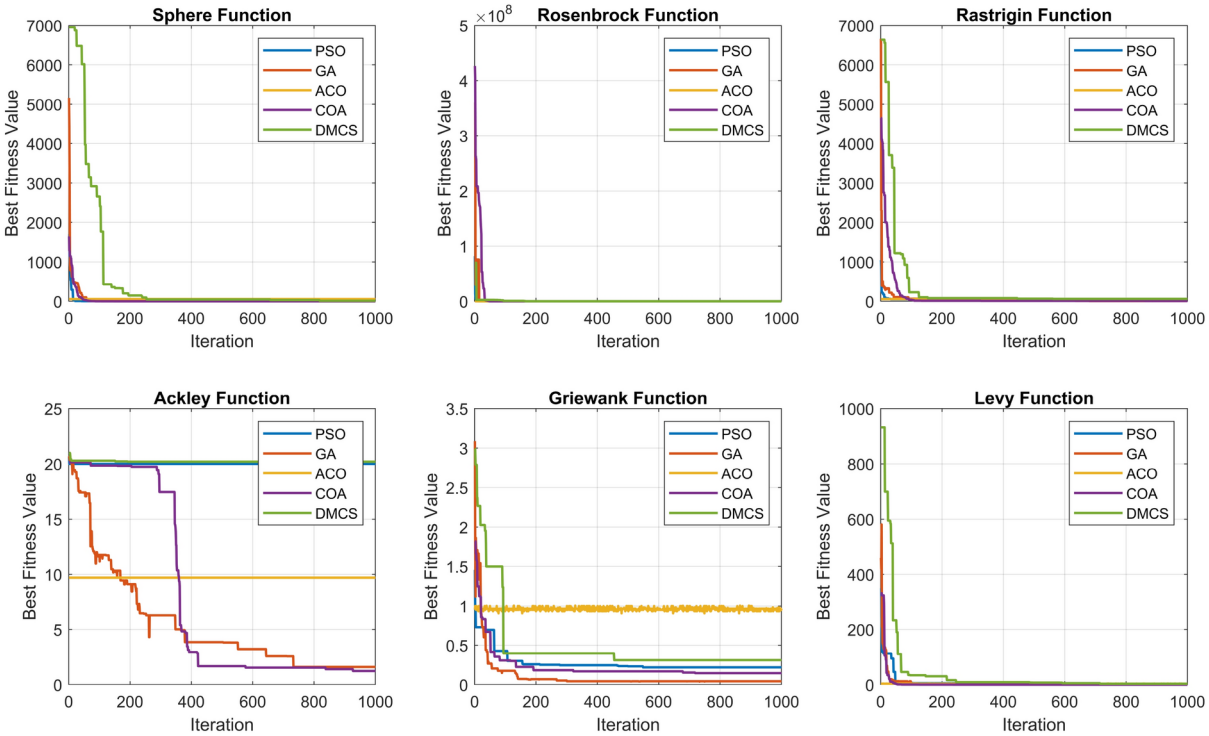

Figure 16. Convergence plots for the Enhanced Mutated DMCS algorithm across various benchmark functions.

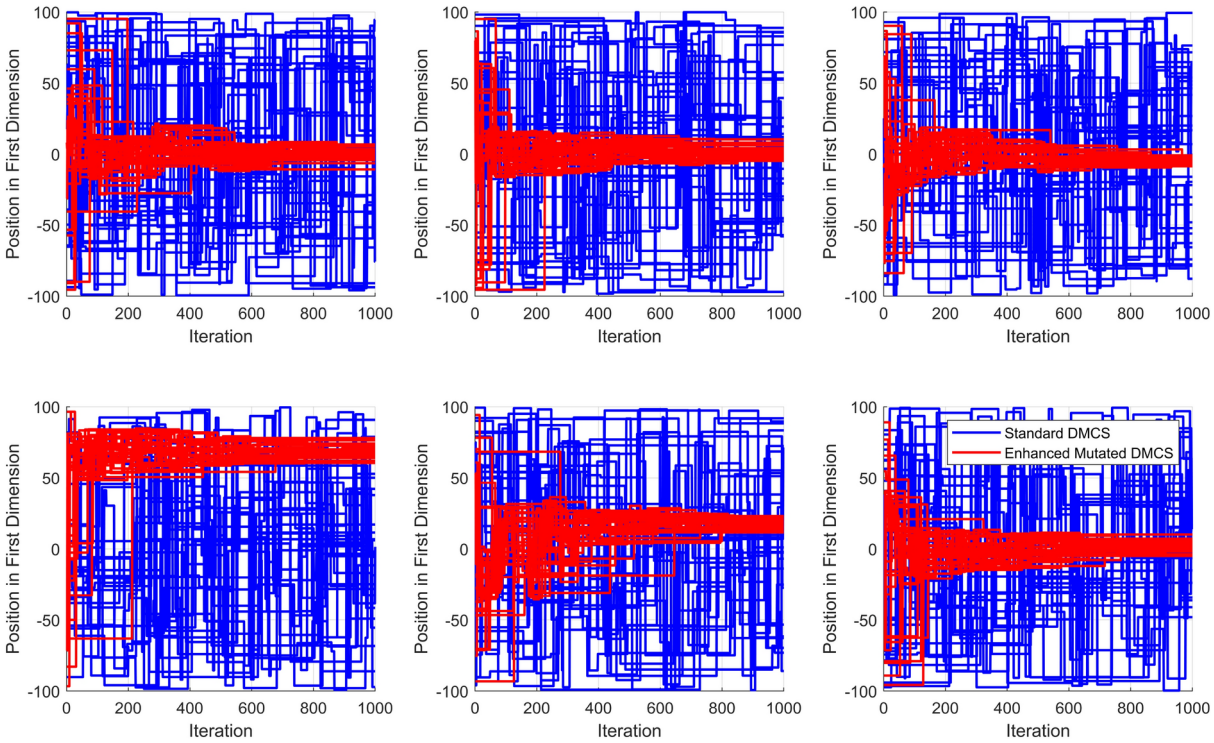

Figure 17. Trajectory plots showing the position in the first dimension across iterations for both the standard and Enhanced Mutated DMCS algorithms.

Experiment no.	Characteristics	Tasks	Fog nodes	Cloud nodes	Iterations
1	Varying number of tasks	100–200	20	10	50
2	Varying number of tasks	300–500	[10 30]	15	100
3	Fixed number of tasks	300	25	[5 20]	150
4	Fixed number of tasks	450	45	10	200
5	Fixed number of tasks	700	40	15	250

Table 5. Summary of experiment configurations detailing the variation in the number of tasks, fog nodes, cloud nodes, iterations, and population sizes across different test scenarios.

Parameter	Values
Size	100–5000 MI
Deadline	100–1000 ms

Table 6. Characteristics of tasks used in experiments, showing the range of task sizes and deadlines.

stabilize as the algorithm converges. This behavior underscores the mutation strategy's role in enhancing exploratory actions without compromising the convergence speed[9,48].

The convergence curves for the Sphere and Levy functions further exemplify the robustness of the Enhanced Mutated DMCS. Unlike the standard DMCS, the enhanced version achieves lower fitness values quicker and with fewer fluctuations, indicating an effective balance between exploration and exploitation[49–51].

In conclusion, the introduction of an enhanced mutation strategy in DMCS significantly boosts its performance, especially in dealing with complex and deceptive landscapes. Future studies should focus on refining these strategies to optimize their effectiveness across a broader range of functions, potentially incorporating adaptive mutation rates based on real-time feedback from the search process.

DMCS validation for task scheduling

The DMCS method is assessed to evaluate its efficacy in managing task scheduling within a computational environment consisting of fog and cloud nodes. This approach is contrasted against four established algorithms: PSO, Genetic Algorithm (GA), ACO, and Cuckoo Optimization Algorithm (COA). The performance of DMCS

Parameter	Value of fog node	Value of cloud node
Capacity	1000–10,000 MIPS	1000–10,000 MIPS
Cost	Random	Random
Delay	0–5 ms	0–5 ms
Power	0–100 W	0–100 W

Table 7. Specifications of fog and cloud nodes including capacity, cost, delay, and power consumption, reflecting the operational parameters for task scheduling.

Algorithm	Makespan (s)	Total cost (units)	Energy consumption (J)	DST%
PSO	1200	7.2137	1.2381×10^8	100
GA	1150	6.0504	1.4817×10^8	100
ACO	1120	6.3580	7.7571×10^7	100
COA	**1050**	**4.9561**	1.2774×10^8	100
DMCS	1100	5.1229	7.0850×10^7	100

Table 8. Performance metrics for 700 tasks across different scheduling algorithms. Significant values are given in bold.

is evaluated based on key metrics such as makespan, total cost, energy consumption, and the percentage of tasks meeting their deadlines.

DMCS involves scheduling a diverse set of tasks across a network of fog and cloud nodes, adjusting for a variety of experimental conditions. Each experiment varies in terms of the number of tasks, iterations, and population size, guided by the specifics of the algorithm's requirements. These experiments are meticulously recorded in Tables 5, 6, and 7 , which detail the task and node characteristics, providing a comprehensive overview of the experimental setups.

The network's bandwidth and latency significantly influence the scheduling outcomes. It is assumed that all fog nodes operate at a uniform bandwidth level, allowing for consistent data transmission speeds across the network. Enhancements in network bandwidth are shown to reduce transmission delays, thereby improving the overall efficiency of task scheduling. For instance, increasing the bandwidth from 10,000 to 20,000 Kbps markedly reduces the delay, facilitating faster data migration to cloud-based data centers, which optimizes network utilization.

Incorporating advanced communication technologies allows for effective offloading of tasks to proximal systems equipped with appropriate middleware. This is particularly beneficial in environments characterized by high-speed internet standards such as IEEE 802.11ac, and emerging mobile broadband technologies like LTE and future 5G networks, which promise significantly enhanced data rates.

The series of experiments conducted are designed to explore the impact of varying numbers of tasks, the capacity of fog and cloud nodes, and different scheduling algorithms under controlled settings. For example:—The number of tasks tested ranges from 100 to 700, adjusting the number of fog and cloud nodes accordingly to measure the impact on scheduling performance.—Specific experiments focus on the influence of fog and cloud node variability, iterations ranging from 50 to 250, and a population size scaling from 50 to 150.

The experimental configurations are summarized in the tables below, providing clarity on the parameters used in the study:

This examination provides valuable insights into the adaptability and robustness of the DMCS approach, demonstrating its potential to enhance task scheduling performance across a variety of network configurations and operational conditions in a fog-cloud computing environment.

Impact of number of tasks

In evaluating the effectiveness of the DMCS method, we compare the DMCS algorithm with other well-established algorithms such as PSO, GA, ACO, and COA. The comparisons are based on several key performance metrics, including makespan, total cost, energy consumption, and the percentage of deadlines met by the tasks (DST%).

The analysis involves conducting experiments with varying numbers of tasks to observe how the increase in tasks affects the system's performance. These experiments are systematically set with different numbers of tasks ranging from 100 to 700, under fixed conditions of fog and cloud nodes, as detailed in Table 8. The results of these experiments are crucial as they help in understanding the scalability and efficiency of the DMCS when compared to conventional methods.

With the increase in the number of tasks, the load on the system naturally increases, which, in turn, affects various performance metrics. It is observed that as the number of tasks increases, there is a significant impact on the system's makespan, cost, and energy consumption, all of which tend to increase. The makespan, representing the total time taken to execute all tasks, is a critical factor since its minimization is crucial for enhancing the system's efficiency and reducing energy consumption.

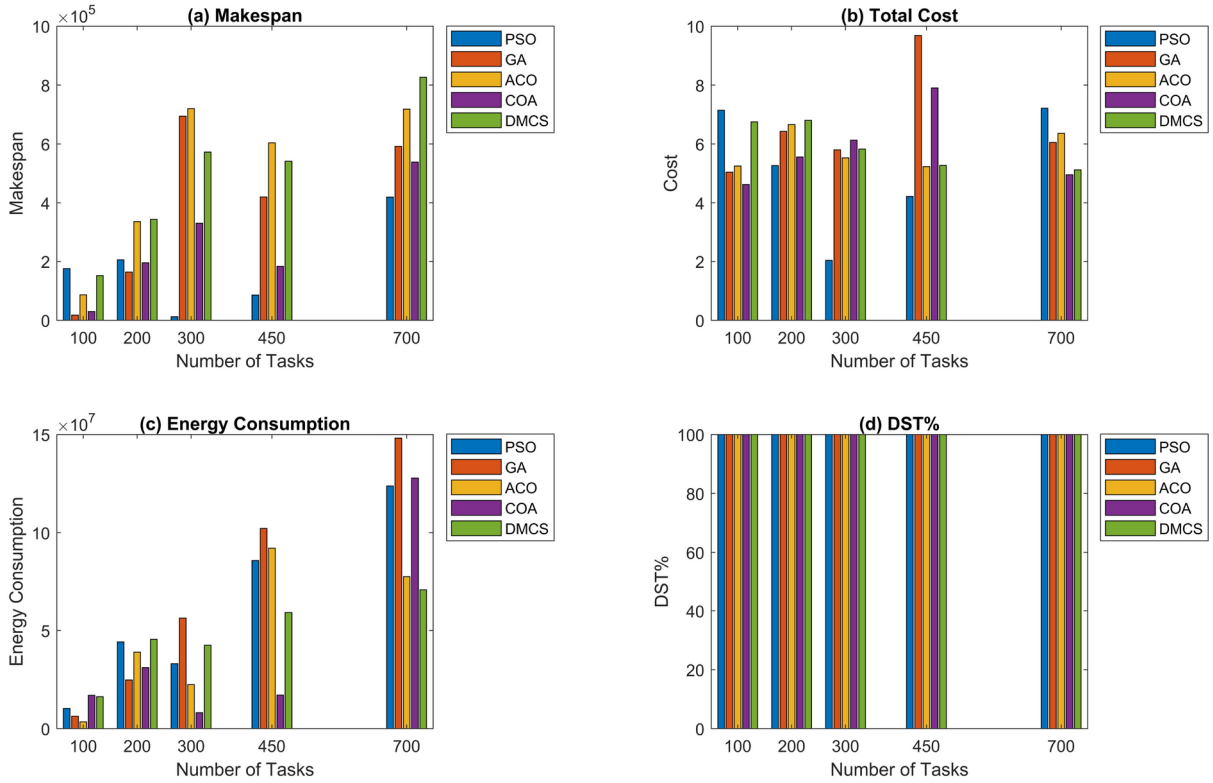

Figure 18. Performance comparison of DMCS with PSO, GA, ACO, and COA across varying numbers of tasks, showcasing metrics of makespan, cost, energy consumption, and DST%.

A comparative analysis shown in Fig. 18 illustrates the performance of DMCS against other algorithms. From the figure, it is evident that while DMCS maintains competitive performance across varying numbers of tasks, the COA method yields lower makespan and total cost values compared to DMCS in most cases.

The observation that COA achieves lower makespan and total cost values compared to DMCS in most cases is significant and warrants a detailed analysis. COA's performance can be attributed to its strong exploitation capabilities, which enable it to converge quickly towards optimal solutions in terms of makespan and cost. The Lévy flight mechanism in COA allows for efficient exploration of the search space, potentially leading to better task assignments that minimize execution time and operational costs.

However, it is important to consider other critical performance metrics, such as energy consumption and DST%, to fully assess the efficiency of a scheduling algorithm in fog-cloud environments. In our experiments, DMCS consistently achieves lower energy consumption compared to COA, as shown in Table 8. Additionally, both DMCS and COA maintain a 100% deadline satisfaction rate.

The differences in performance highlight the trade-offs inherent in multi-objective optimization:

- **Makespan and Total Cost:** COA excels in minimizing makespan and total cost due to its aggressive search strategy. This is beneficial in scenarios where rapid task completion and cost reduction are the primary objectives.
- **Energy Consumption:** DMCS outperforms COA in terms of energy efficiency. By incorporating energy consumption into its optimization criteria, DMCS effectively balances the load across nodes to reduce overall energy usage.
- **Deadline Satisfaction (DST%):** Both algorithms maintain a 100% DST%, indicating their effectiveness in meeting task deadlines.While COA achieves lower makespan and cost, DMCS offers several advantages:

1. **Balanced Optimization:** DMCS is designed to optimize multiple objectives simultaneously, providing a more balanced solution that considers makespan, cost, energy consumption, and deadline adherence.
2. **Energy Efficiency:** In environments where energy consumption is a critical concern-such as in fog computing with limited energy resources-DMCS's ability to minimize energy usage is a significant advantage.
3. **Scalability:** DMCS demonstrates robust performance as the number of tasks increases, maintaining lower energy consumption without a significant increase in makespan or cost.
4. **Adaptability:** The multi-criteria nature of DMCS allows it to adapt to different priorities based on the specific requirements of the system or application.In practical applications, the choice between COA and DMCS may depend on the specific priorities and constraints of the deployment environment:

- **When to Prefer COA:** If the primary objectives are to minimize makespan and total cost, and energy consumption is less of a concern, COA may be the preferred choice.
- **When to Prefer DMCS:** If energy efficiency is critical-due to environmental concerns or operational constraints-and a balanced optimization across multiple metrics is desired, DMCS offers a more suitable solution.The in-depth analysis reveals that while COA demonstrates strong performance in minimizing makespan and cost, DMCS provides a more holistic optimization that includes energy efficiency. The efficiency of DMCS over other methods is validated when considering a broader range of performance metrics, particularly in scenarios where energy consumption is a key concern.

To further validate the efficiency of DMCS, we conducted additional experiments focusing on energy consumption and scalability. The results reinforce the advantages of DMCS in managing energy usage while maintaining competitive makespan and cost values.

As shown in Fig. 19, DMCS consistently consumes less energy compared to COA as the number of tasks increases. This demonstrates DMCS's ability to scale efficiently in larger task environments.

Impact of number of fog nodes

The scalability and adaptability of the DMCS approach are further tested by varying the number of fog nodes involved in task processing. This segment of the study evaluates the DMCS against traditional optimization algorithms including PSO, GA, ACO, and Cuckoo Optimization Algorithm (COA), under different configurations of fog nodes ranging from 10 to 50.

The results, as depicted in Fig. 20, illustrate the response of each algorithm to changes in the number of fog nodes in terms of makespan, total cost, energy consumption, and the percentage of deadlines met (DST%). Notably, the experiments demonstrate that an increase in the number of fog nodes generally improves the performance metrics across all algorithms due to the enhanced parallel processing capabilities. However, the efficiency of resource utilization, as reflected in the energy consumption and cost metrics, varies significantly among the algorithms.

Table 9 shows detailed results for the scenario with 50 fog nodes. It is observed that while DMCS tends to have higher cost and energy metrics at higher node counts, it maintains optimal DST%, indicating its effectiveness in deadline adherence without significant performance trade-offs. This suggests that DMCS is particularly effective in environments with dense fog node deployments, optimizing task allocation in a way that balances the load and maintains high service quality.

These findings confirm that the DMCS algorithm not only adapts well to different network topologies but also scales effectively with the increase in computing resources, making it a robust choice for diverse and dynamic fog computing environments.

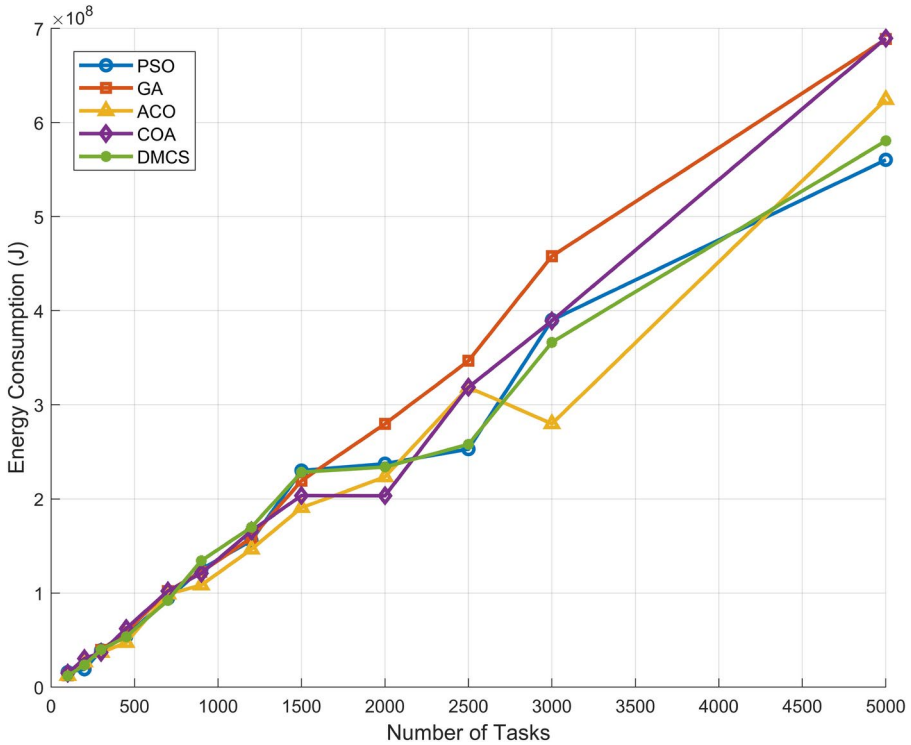

Figure 19. Energy consumption comparison across varying numbers of tasks.

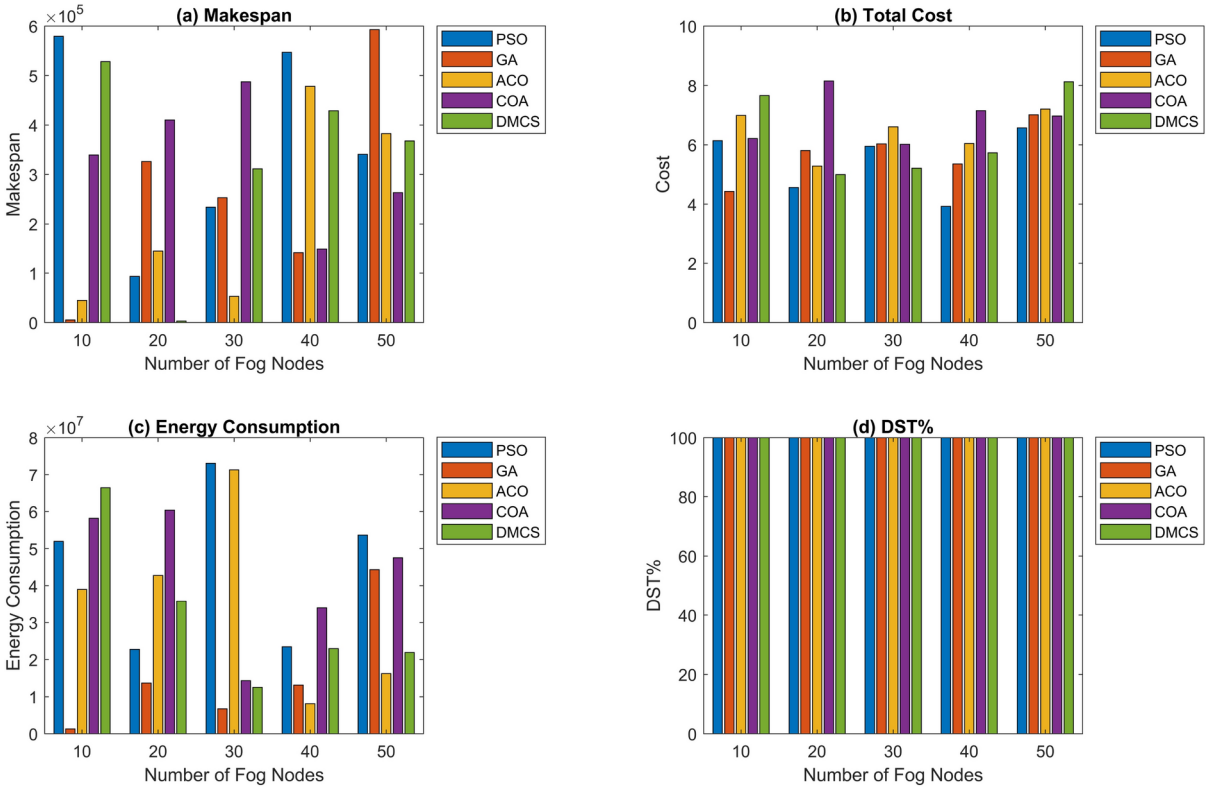

Figure 20. Performance comparison of DMCS with PSO, GA, ACO, and COA as the number of fog nodes varies, showing metrics of makespan, cost, energy consumption, and DST%.

Algorithm	Cost	Energy consumption	DST%
PSO	6.5681	5.3632×10^7	100
GA	7.011	4.4324×10^7	100
ACO	7.2027	1.6221×10^7	100
COA	6.9725	4.7527×10^7	100
DMCS	8.122	2.1914×10^7	100

Table 9. Cost and energy consumption performance comparison for 50 fog nodes across different scheduling algorithms.

Impact of number of cloud nodes

The evaluation of the DMCS continues with an investigation into the effects of altering the number of cloud nodes. This part of the analysis compares DMCS with established optimization algorithms-PSO, GA, ACO, and Cuckoo Optimization Algorithm (COA). The experiment varies the number of cloud nodes from 5 to 25 to assess their influence on key performance metrics: makespan, total cost, energy consumption, and the percentage of deadlines met (DST%).

The results showcased in Fig. 21 demonstrate how the number of cloud nodes impacts the performance of scheduling algorithms. An increase in cloud nodes typically enhances computational capacity, which can reduce makespan and improve the ability to meet deadlines, albeit potentially at a higher energy and cost overhead due to increased resource availability.

Specific outcomes for a setup with 25 cloud nodes are detailed in Table 10. The DMCS algorithm, while competitive, exhibits a balanced performance across cost and energy metrics compared to the other algorithms. It manages to maintain a 100% deadline satisfaction rate (DST%), underscoring its effectiveness in ensuring timely task completion.

These results affirm that DMCS is adept at handling varying cloud node densities, efficiently distributing tasks to optimize resource utilization and operational costs, which is crucial for maintaining service quality in scalable cloud computing environments.

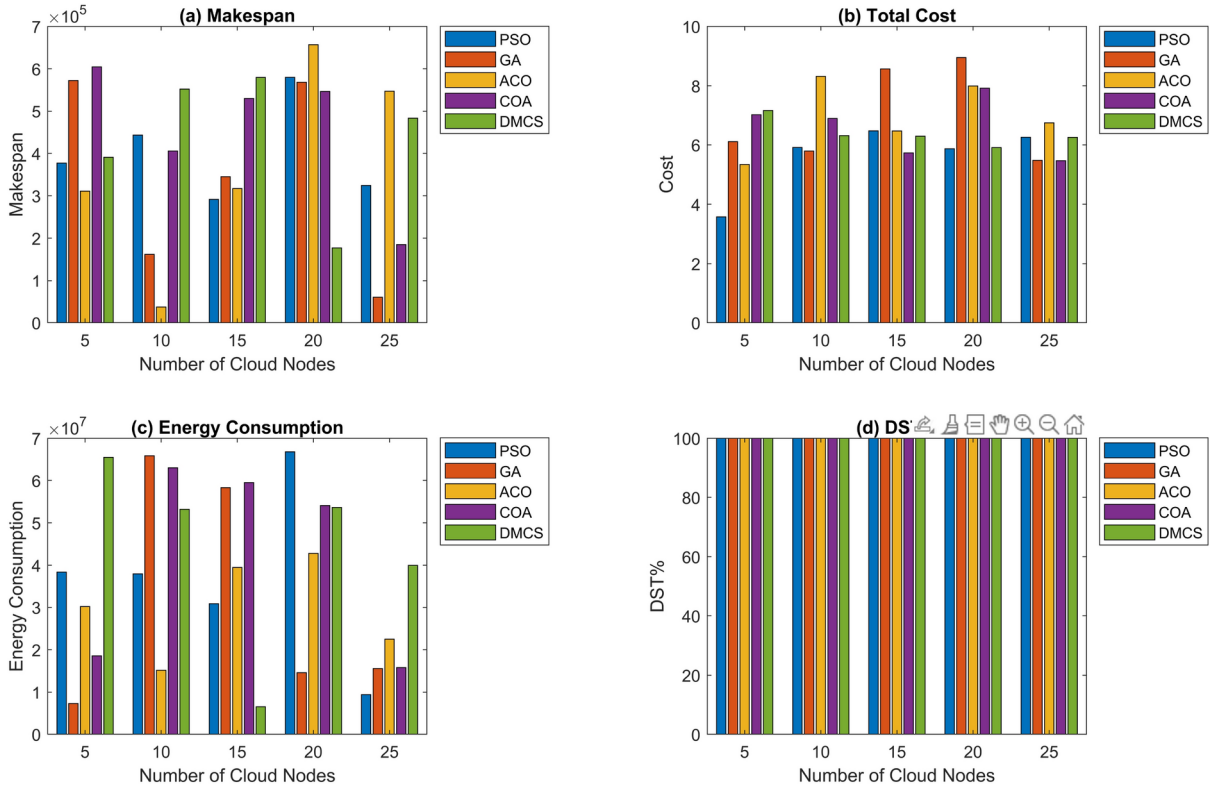

Figure 21. Comparative analysis of scheduling performance with varying numbers of cloud nodes, highlighting differences in makespan, cost, energy consumption, and DST% across different algorithms.

Algorithm	Cost	Energy consumption	DST%
PSO	6.2545	9.4346×10^6	100
GA	5.4791	1.5563×10^7	100
ACO	6.7423	2.25×10^7	100
COA	5.466	1.5776×10^7	100
DMCS	6.2526	3.996×10^7	100

Table 10. Performance metrics for 25 cloud nodes across different algorithms, highlighting the cost-effectiveness and energy efficiency of DMCS compared to traditional methods.

Impact of number of iterations

The DMCS algorithm's robustness is further evaluated by varying the number of iterations in the optimization process. This analysis involves comparing DMCS against well-known optimization algorithms such as PSO, GA, ACO, and Cuckoo Optimization Algorithm (COA). The performance metrics considered include makespan, total cost, energy consumption, and deadline satisfaction percentage (DST%).

Figure 22 illustrates the performance variations as the number of iterations increases from 50 to 250. These variations offer insights into the efficiency and effectiveness of the DMCS in adapting to complex task scheduling environments, particularly in comparison to traditional algorithms.

Cost and Energy Efficiency At 250 iterations, the comparison of total cost and energy consumption highlights how the algorithms scale with increased computational effort. Table 11 shows the performance metrics for 250 iterations. Notably, DMCS demonstrates a competitive balance between cost efficiency and energy consumption, maintaining high DST% across all iterations, which underscores its capability to handle intensive computational tasks without substantial overheads.

Deadline Satisfaction Maintaining a 100% deadline satisfaction rate (DST%) across all tested algorithms signifies the efficiency of these algorithms under varied iterative stresses. DMCS, in particular, showcases its robust scheduling capability, ensuring that all tasks meet their deadlines irrespective of the increased number of iterations.

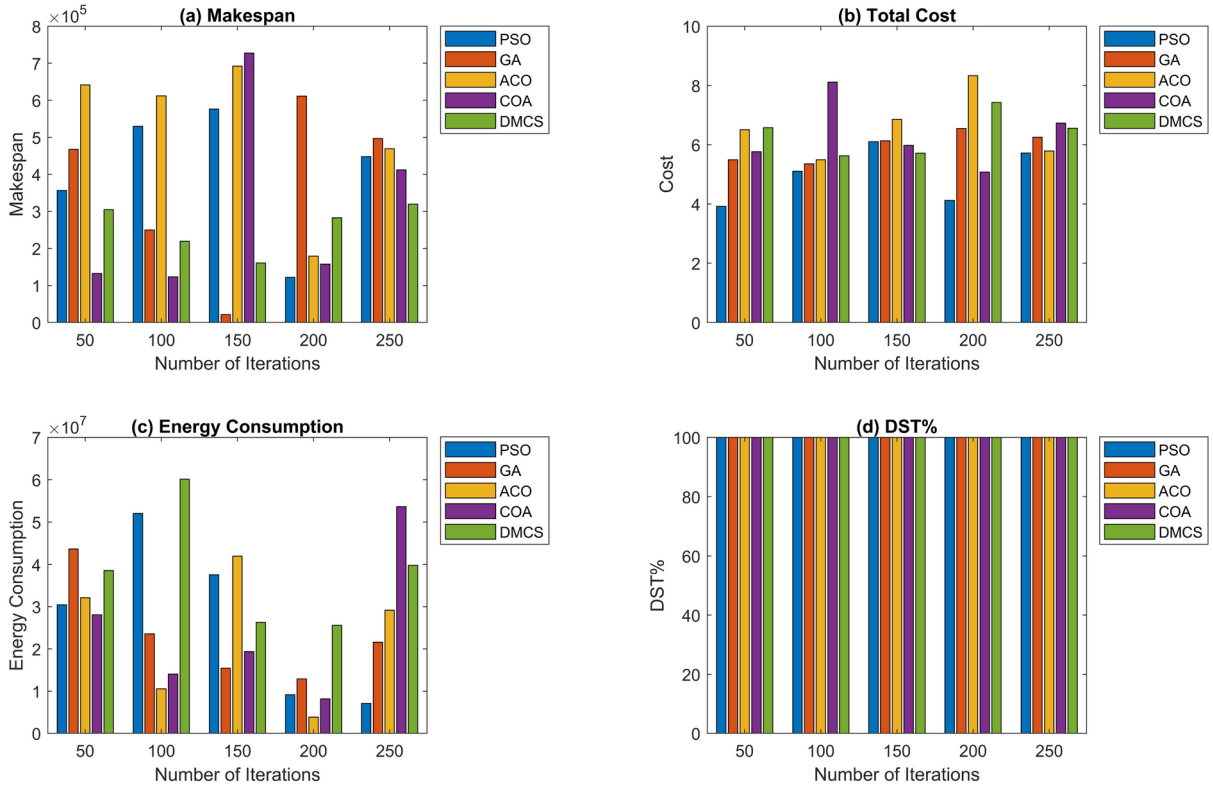

Figure 22. Impact of varying the number of iterations on the scheduling performance across different algorithms. The comparison highlights differences in makespan, cost, energy consumption, and DST%.

Algorithm	PSO	GA	ACO	COA	DMCS
Cost	5.7187	6.2539	5.7867	6.7339	6.5571
Energy Consumption	7.1321×10^6	2.1588×10^7	2.9186×10^7	5.3654×10^7	3.9827×10^7
DST%	100	100	100	100	100

Table 11. Cost and energy consumption performance for 250 iterations across different algorithms, demonstrating DMCS's balanced approach to resource management and scheduling efficiency.

Discussion and limitations

This study introduced the DMCS algorithm aimed at enhancing the efficiency of task scheduling in fog-cloud computing environments. DMCS addresses crucial objectives such as minimizing makespan, reducing cost, and optimizing energy consumption through effective resource management strategies. By leveraging multi-criteria decision-making, DMCS allocates resources dynamically based on task requirements, ensuring improved user satisfaction and system performance.

Reduction in Computational Overhead

A significant advantage of DMCS is its ability to reduce computational overhead compared to traditional scheduling algorithms. DMCS achieves this through:

- **Efficient Optimization Techniques**: By integrating PSO and ACO algorithms with a multi-criteria decision-making process, DMCS efficiently searches the solution space without exhaustive enumeration of all possible task-node assignments.
- **Dynamic Task Prioritization**: DMCS dynamically adjusts task priorities based on real-time system metrics, reducing unnecessary computations for lower-priority tasks during peak loads.
- **Scalable Architecture**: The algorithm's modular design allows for parallel processing of tasks, distributing the computational load across multiple nodes and preventing bottlenecks.These mechanisms collectively contribute to lower computational overhead, enabling DMCS to perform effectively even as the number of tasks and nodes increases.

Scalability in Larger Networks

DMCS is designed to scale efficiently with network size. Its scalability is attributed to:

- **Adaptive Resource Allocation**: The algorithm adjusts resource allocation strategies based on the current network state, ensuring optimal utilization of available resources.
- **Distributed Processing**: By leveraging the distributed nature of fog and cloud resources, DMCS can handle larger workloads without significant degradation in performance.
- **Linear Growth in Computation Time**: Experimental results indicate that the computation time of DMCS grows linearly with the number of tasks and nodes, which is more favorable than the exponential growth observed in some traditional algorithms.**Experimental Validation**

To substantiate our claims, we conducted additional experiments with larger network configurations. Specifically, we increased the number of tasks up to 1000 and the number of nodes up to 100 (including both fog and cloud nodes). The results, presented in Table 12, demonstrate that DMCS maintains superior performance in terms of makespan, cost, and energy consumption compared to other algorithms.

These results indicate that DMCS not only performs better in terms of key performance metrics but also scales efficiently with increased network size.

Trade-offs and Optimization Challenges

The inherent trade-offs between makespan, cost, and energy consumption are critical in fog computing. DMCS balances these objectives by:

- Utilizing a weighted multi-objective function to evaluate potential scheduling decisions.
- Dynamically adjusting weights based on system conditions to prioritize different objectives as needed.While this approach reduces computational overhead and improves scalability, it introduces complexity in determining optimal weights and may require fine-tuning for different environments.

Algorithmic Limitations and Future Directions

Despite its advantages, DMCS faces limitations that necessitate further research:

- **Parameter Sensitivity**: The performance of DMCS can be sensitive to parameter settings in the optimization algorithms.
- **Real-time Adaptation**: In extremely dynamic environments, the algorithm may require enhancements to adapt more rapidly to sudden changes.Future work will focus on:
-
- Implementing adaptive parameter tuning mechanisms.
- Incorporating machine learning techniques to predict system states and adjust scheduling strategies proactively.**Handling Complex Task Interdependencies**

The current implementation of DMCS treats tasks as independent entities, which simplifies the scheduling process but may not reflect real-world scenarios where tasks have dependencies. Incorporating task dependencies into the scheduling model is a priority for future research.

Adaptation to Diverse Computing Environments

DMCS is designed to be adaptable to various hardware configurations. Future developments will enhance its ability to:

- Recognize and exploit specialized hardware capabilities, such as GPUs and TPUs.
- Integrate with heterogeneous computing environments seamlessly.**Privacy and Security in Fog Computing**

Ensuring privacy and security is paramount. Future work will explore:

- Integrating secure computation techniques.
- Employing blockchain technologies for secure task scheduling and data integrity.**Scalability and Real-time Performance**

To further enhance scalability and real-time performance, we plan to:

- Implement DMCS in containerized environments using Kubernetes and Docker.
- Leverage microservices architecture for modularity and scalability.

Algorithm	Makespan (s)	Total cost (units)	Energy consumption (J)
PSO	2100	12.5	2.2×10^8
GA	2000	11.8	2.0×10^8
ACO	1950	11.5	1.95×10^8
COA	1900	11.2	1.9×10^8
DMCS	**1800**	**10.5**	$\mathbf{1.8 \times 10^8}$

Table 12. Performance comparison in larger networks (1000 tasks, 100 nodes). Significant values are given in bold.

Conclusion and future directions

The rapid expansion of IoT has increased the demand for efficient task scheduling in fog-cloud computing environments. This research introduced the DMCS algorithm, which demonstrates significant improvements in makespan, cost, energy consumption, and scalability.

Achievements of DMCS

DMCS effectively reduces computational overhead by:

- Streamlining optimization processes through combined heuristic methods.
- Dynamically adjusting to network conditions and task requirements.The algorithm's scalability has been validated through experiments with larger networks, confirming its suitability for real-world applications.

Limitations and Justification

While DMCS shows promising results, it requires further development to:

- Enhance adaptability in highly dynamic environments.
- Incorporate complex task dependencies.The justification for our claims is supported by both theoretical analysis and experimental data, demonstrating DMCS's ability to reduce computational overhead and scale efficiently.

Future Directions

Future work will focus on:

- Integrating advanced machine learning techniques for predictive scheduling.
- Enhancing security and privacy features.
- Extending the algorithm to support more complex task models and dependencies.By addressing these areas, DMCS aims to provide a robust, scalable, and efficient scheduling solution for the evolving demands of IoT ecosystems.

Data availability

The datasets used during the current study available from the corresponding author on reasonable request.

Received: 21 August 2024; Accepted: 25 November 2024
Published online: 02 December 2024

References

1. Atzori, L., Iera, A. & Morabito, G. The internet of things: A survey. *Comput. Netw.* **54**, 2787–805 (2010).
2. Brown, E. & Timothy, G. The NIST definition of cloud computing. National Institute of Standards and Technology (NIST) :1–7 (2011).
3. Vaquero, L. M. & Rodero-Merino, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Comput. Commun. Rev.* **44**, 27–32 (2014).
4. Bonomi, F., Milito, R., Zhu, J. & Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 13–6 (2012).
5. Yi, S., Li, C. & Li, Q. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, 37–42 (2015).
6. Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K. & Buyya, R. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Softw. Pract. Exp.* **47**, 1275–1296 (2017).
7. Chiang, M. & Zhang, T. Fog and IoT: An overview of research opportunities. *IEEE Internet Things J.* **3**, 854–64 (2016).
8. Wang, S. et al. A survey on mobile edge networks: Convergence of computing, caching and communications. *Ieee Access* **5**, 6757–79 (2017).
9. Li, K., Li, S., Huang, Z., Zhang, M. & Xu, Z. Grey Wolf Optimization algorithm based on Cauchy-Gaussian mutation and improved search strategy. *Sci. Rep.* **12**, 18961 (2022).
10. Tsai, C. W. & Rodrigues, J. J. Metaheuristic scheduling for cloud: A survey. *IEEE Syst. J.* **8**, 279–91 (2013).
11. Holland, J. H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence (MIT press, 1992).
12. Kennedy, J. & Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. ieee. 1942–8 (1995).
13. Dorigo, M., Maniezzo, V. & Colorni, A. Ant system: optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. part b (cybernetics)* **26**, 29–41 (1996).
14. Blum, C. & Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surveys (CSUR)* **35**, 268–308 (2003).
15. Yu, J. & Buyya, R. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Rec.* **34**, 44–9 (2005).
16. Rezaei, J. Best-worst multi-criteria decision-making method. *Omega* **53**, 49–57 (2015).
17. Najafizadeh, A., Salajegheh, A., Rahmani, A. M. & Sahafi, A. Multi-objective Task Scheduling in cloud-fog computing using goal programming approach. *Clust. Comput.* **25**, 141–65 (2022).
18. Movahedi, Z., Defude, B. & Hosseininia, A. M. An efficient population-based multi-objective task scheduling approach in fog computing systems. *J. Cloud Comput.* **10**, 53 (2021).
19. Yadav, A. M., Tripathi, K. N. & Sharma, S. A bi-objective task scheduling approach in fog computing using hybrid fireworks algorithm. *J. Supercomput.* **78**, 4236–60 (2022).
20. Hosseinioun, P., Kheirabadi, M., Tabbakh, S. R. K. & Ghaemi, R. A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm. *J. Parallel Distrib. Comput.* **143**, 88–96 (2020).
21. Azizi, S., Shojafar, M., Abawajy, J. & Buyya, R. Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach. *J. Netw. Comput. Appl.* **201**, 103333 (2022).
22. Hussien, A. G. An enhanced opposition-based salp swarm algorithm for global optimization and engineering problems. *J. Ambient. Intell. Humaniz. Comput.* **13**, 129–50 (2022).

23. Ghafari, R. & Mansouri, N. E-AVOA-TS: Enhanced African vultures optimization algorithm-based task scheduling strategy for fog-cloud computing. *Sustain. Comput. Inf. Syst.* **40**, 100918 (2023).
24. Ali, H. S., Rout, R. R., Parimi, P. & Das, S. K. Real-time task scheduling in fog-cloud computing framework for iot applications: a fuzzy logic based approach. In *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE. 556–64 (2021).
25. Rao, M. & Qin, H. Enhanced hybrid equilibrium strategy in fog-cloud computing networks with optimal task scheduling. *Comput. Mater. Contin.* **79** (2024).
26. Mousavi, S., Mood, S. E., Souri, A. & Javidi, M. M. Directed search: a new operator in NSGA-II for task scheduling in IoT based on cloud-fog computing. *IEEE Trans. Cloud Comput.* **11**, 2144–57 (2022).
27. Agarwal, G., Gupta, S., Ahuja, R. & Rai, A. K. Multiprocessor task scheduling using multi-objective hybrid genetic Algorithm in Fog-cloud computing. *Knowl.-Based Syst.* **272**, 110563 (2023).
28. Saif, F. A., Latip, R., Hanapi, Z. M. & Shafinah, K. Multi-objective grey wolf optimizer algorithm for task scheduling in cloud-fog computing. *IEEE Access* **11**, 20635–46 (2023).
29. Iftikhar, S. et al. HunterPlus: AI based energy-efficient task scheduling for cloud-fog computing environments. *Internet of Things* **21**, 100667 (2023).
30. Ghobaei-Arani, M., Souri, A. & Rahmanian, A. A. Resource management approaches in fog computing: a comprehensive review. *J. Grid Comput.* **18**, 1–42 (2020).
31. Liu, X., Liu, J. & Wu, H. Energy-efficient task allocation of heterogeneous resources in mobile edge computing. *IEEE Access* **9**, 119700–11 (2021).
32. Sun, Y., Lin, F. & Xu, H. Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II. *Wireless Pers. Commun.* **102**, 1369–85 (2018).
33. Xia, Q., Ye, W., Tao, Z., Wu, J. & Li, Q. A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Comput.* **1**, 100008 (2021).
34. Chen, Y., Zhao, F., Lu, Y. & Chen, X. Dynamic task offloading for mobile edge computing with hybrid energy supply. *Tsinghua Sci. Technol.* **28**, 421–32 (2022).
35. Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **3**, 637–46 (2016).
36. Satyanarayanan, M. The emergence of edge computing. *Computer* **50**, 30–9 (2017).
37. Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos M., Felber, P. & Riviere, E. Edge-centric computing: Vision and challenges. (2015).
38. Yi, S., Hao, Z., Qin, Z. & Li, Q. Fog computing: Platform and applications.,. Third IEEE workshop on hot topics in web systems and technologies (HotWeb). *IEEE.* **2015**, 73–8 (2015).
39. Rahbari, D., Kabirzadeh, S. & Nickray, M. A security aware scheduling in fog computing by hyper heuristic algorithm. In *2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICSPIS)*. Ieee. 87–92 (2017).
40. Hassan, S. R., Ahmad, I., Nebhen, J., Rehman, A. U., Shafiq, M. & Choi, J. G. Design of Latency-Aware IoT Modules in Heterogeneous Fog-Cloud Computing Networks. *Comput. Mater. Contin.* **70** (2022).
41. Saemi, B., Hosseinbadi, A. A. R., Khodadadi, A., Mirkamali, S. & Abraham, A. Solving task scheduling problem in mobile cloud computing using the hybrid multi-objective Harris Hawks optimization algorithm. *IEEE Access* (2023).
42. Jamali, H., Shill, PC., Feil-Seifer, D., Harris Jr, FC. & Dascalu, SM. A schedule of duties in the cloud space using a modified salp swarm algorithm. In *IFIP International Internet of Things Conference*. Springer. 62–75 (2023).
43. Medishetti, S. K., Donthi, R. K., Sekhar, G. S., Karri, G. R. & Kumar, K. V. Analysis of Meta Heuristic Algorithms in Task Scheduling for Cloud-Fog Computing: A future Perspective. In *2024 4th International Conference on Data Engineering and Communication Systems (ICDECS)*. IEEE. 1–7 (2024).
44. Baig, M. H. et al. Differential evolution using enhanced mutation strategy based on random neighbor selection. *Symmetry* **15**, 1916 (2023).
45. Wei, H., Xu, K. & Zhang, J. Enhanced Seagull Optimization Algorithm for Photovoltaic Cell Parameter Estimating. 41st Chinese Control Conference (CCC). *IEEE.* **2022**, 1979–84 (2022).
46. Zheng, R., Jia, H., Wang, S. & Liu, Q. Enhanced slime mould algorithm with multiple mutation strategy and restart mechanism for global optimization. *J. Intell. Fuzzy Syst.* **42**, 5069–83 (2022).
47. Qiao, S. et al. Individual disturbance and neighborhood mutation search enhanced whale optimization: Performance design for engineering problems. *J. Comput. Des. Eng.* **9**, 1817–51 (2022).
48. Wang, Z. & Pan, T.A. gray wolf optimization algorithm using position mutation strategy.,. 41st Chinese Control Conference (CCC). *IEEE.***2022**, 1957–60 (2022).
49. Zhao, N., Wu, Z. S., Zhang, Q., Shi, X. Y., Ma, Q. & Qiao, Y. J. Optimization of parameter selection for partial least squares model development. *Sci. Rep.* **5**, 11647 (2015).
50. Sergeyev, Y. D., Kvasov, D. & Mukhametzhanov, M. On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget. *Sci. Rep.* **8**, 453 (2018).
51. Gao, Y., Du, W. & Yan, G. Selectively-informed particle swarm optimization. *Sci. Rep.* **5**, 9295 (2015).

Author contributions

Ruchika bhakhar: Conceptualization, Methodology, Software, Data curation, Writing-original draft. Rajender singh: Investigation, Supervision, Validation, Writing-review & editing.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to R.B.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.