INTERNATIONAL JOURNAL OF Telecommunications
AIR DEFENSE COLLEGE

Profit Maximization of Offloading Tasks for Mobile Edge Computing in C-RAN Using Ant Colony Algorithm

Citation: Aboul, S.; Abd El Kader, M., Eid E., Ali, S.
Inter. Jour. of Telecommunications, ***IJT'2025****, Vol. 05, Issue 02, pp. 1-23, 2025.*
Doi: 10.21608/ijt.2025.399953.1123
Editor-in-Chief: Youssef Fayed.
Received: 29/07/2025.
Accepted date: 10/09/2025.
Published date: 10/09/2025.

Sherif M. Aboul [1*], Hala M.Abd El Kader[2], Esraa M.Eid[3], Shimaa S.Ali[4].
*[1]Communication Department, Faculty of Engineering at Shoubra, Benha University, Electrical Department, Faculty of Engineering, MTI University, Cairo, Egypt. s.elsayed69233@feng.bu.edu.eg
[2]Communication Department, Faculty of Engineering at Shoubra, Benha University, Cairo, Egypt. hala.mansour@feng.bu.edu.eg
[3]Communication Department, Faculty of Engineering at Shoubra, Benha University, Faculty of Computer Science, Benha National University, Cairo, Egypt. esraa.soliman@feng.bu.edu.eg
[4]Communication Department, Faculty of Engineering at Shoubra, Benha University, Cairo, Egypt. shimaa.salama@feng.bu.edu.eg

Abstract: Mobile Edge Computing (MEC) is vital for next-generation low-latency services, enabling resource-constrained mobile devices to offload intensive tasks to cloud-based infrastructure. This reduces energy consumption and latency, making MEC a key component of future mobile networks. Utilizing the Cloud-Radio Access Network (C-RAN) architecture, which integrates Baseband-Units (BBU) with MEC servers and Remote-Radio-Heads (RRHs), complex tasks are executed closer to users, improving service quality and creating new revenue streams for network operators. This paper examines computational offloading profitability from a network operator's perspective. The offloading process involves optimizing radio and computational resources, posing a non-deterministic polynomial-time (NP) hard problem. To address this complexity, four optimization algorithms are evaluated: Ant-Colony Optimization (ACO), Normal-Genetic-Algorithm (NGA), Fast-Genetic-Algorithm (FGA), and Modified Spectrum Efficiency-Based Joint Optimization for Offloading and Resource-Allocation (Modified SJOORA). ACO, inspired by ant behavior, seeks optimal paths, while NGA and FGA simulate natural selection, with FGA offering faster convergence. Modified SJOORA is designed to optimize resource-allocation in MEC environments. The study compares these algorithms under various conditions to identify the most effective for profit maximization. Additionally, a novel approach reduces computational time by using a strategically seeded population and regression-based machine learning to estimate resource-allocation, maintaining accuracy while enhancing efficiency.

Keywords: Profit Maximization; Resource Allocation; Cloud Radio Access Network; Mobile Edge Computing; Optimization Algorithms.

1. Introduction

The increasing computational demands on mobile devices, particularly from advanced multimedia applications like online gaming, present significant challenges for the industry. As these applications require ever-greater processing power, they contribute to faster battery depletion in user equipment (UE). However, consistent anual advancements in battery capacity offer some mitigation for these challenges [1]. Mobile computing is presently challenged by two major issues: narrow wireless bandwidth and insufficient battery capacity. These

limitations influence how systems are architected, how efficiently they operate, and how end-users interact with their devices and digital services. A promising approach to manage energy constraints is offloading, which can effectively help reduce energy consumption [2]. Two primary offloading strategies have been identified. The first involves an UE decision-making process to determine Whether task offloading offers energy savings can be evaluated by contrasting the power consumption of executing tasks locally with that of transmitting the data to an edge computing server [3]. A second method adopts a centralized controller that utilizes Channel-State-Information (CSI) and comprehensive task details from all associated devices, and employs a task-partitioning framework to select UEs for task offloading [3]. This controller also integrates a joint optimization approach to manage energy consumption, delay, and additional resource constraints [4,5]. Cloud-Radio Access Network (C-RAN) and Mobile-Edge-Computing (MEC) are two emerging cloud technologies expected to be essential in supporting future services for UEs. The C-RAN framework adopts a distributed architecture, where a centralized Baseband Unit (BBU) pool is paired with several Remote-Radio-Heads (RRHs) to manage radio access functions. In contrast, MEC is designed to provide computational resources closer to the network edge, thereby enhancing processing capabilities near UEs. Together, these solutions support the delivery of advanced applications and services to mobile devices. In a typical C-RAN setup, the main computational tasks are handled within the centralized BBU pool, while RRHs act as intermediaries, transmitting signals between BBUs and mobile users over the radio spectrum, with bidirectional communication. This setup allows for efficient interference management and cancellation through coordinated processing among BBU pools [6]. Additionally, UEs can offload computation-heavy tasks to nearby edge servers [7], which are generally situated near the BBU pool. By transferring computational workloads to external edge resources, this approach reduces energy usage on mobile devices. However, as the network grows in size, maintaining energy efficiency becomes more computationally intensive. Consequently, a distributed multi-cell MEC optimization strategy is being considered to address these demands [8]. MEC servers, deployed at the edge of the mobile network within Base Stations (BSs) [9], utilize the computational and storage resources of the BBU pool. This setup benefits from the close proximity of RRHs to mobile users within a C-RAN framework. Previous studies have examined the interaction and combined potential of MEC and C-RAN architectures, revealing promising synergies for enhanced service efficiency and resource management.

ACO is a bio-inspired metaheuristic algorithm that mimics the foraging behavior of ants to solve optimization problems. It relies on pheromone trails to iteratively identify and refine high-quality solutions. This approach has been widely adopted in computational systems for tackling resource-allocation, scheduling, and routing tasks due to its efficiency and adaptability. In 5G C-RAN environments, ACO effectively addresses the challenges of resource-allocation and task scheduling by optimizing user demand distribution across network components. The algorithm operates by modeling resource-allocation as a multi-objective optimization problem, balancing network performance metrics like execution time and profit. Ants explore potential solutions based on probabilistic transitions guided by pheromone intensity and heuristic information, which are influenced by factors like computational capacity and network latency. ACO's capacity to respond flexibly to changing environments makes it especially effective for handling large-scale optimization tasks in edge computing and 5G systems. Its integration with machine learning and hybrid optimization methods further enhances its potential to meet stringent real-time requirements, ensuring efficient resource utilization and reduced latency. This capability positions ACO as a critical tool for optimizing task processing in modern, time-sensitive communication systems [10].

where Yuyi et al. [2] present a detailed review of MEC, with a focus on optimizing resource-allocation and tackling computational challenges associated with energy efficiency and latency reduction [11]. Their work encompasses both theoretical advancements and practical methodologies. Vambe et al. [12] introduced an MC-RAN-based solution aimed at enhancing the assignment of virtual machines (VMs) to reduce energy use in a mobile cloning environment. He et al. [13] presented an approach involving the advance loading of task data to lower power consumption during the processing of client operations. Acheampong et al. [14] integrated C-RAN with MEC, focusing on the power-performance tradeoff to enhance Mobile Service Provider (MSP) revenue through coordinated resource-allocation, while acknowledging associated cost constraints Lei Pan et al. [15] explored a range of evolutionary and multi-objective approaches for MEC offloading, incorporating a

genetic algorithm enhanced by Taguchi optimization techniques. This study introduces the Multi-Objective Clustering Evolutionary Algorithm (MCEA), which effectively optimizes cost, energy consumption, and deadline adherence, demonstrating superior performance compared to previous approaches. Tunga et al. [10] produces the integration of ACO with the C-RAN model. It focuses on using ACO for optimizing task offloading in MEC to maximize profit and minimize latency. Ge et al. [16] introduce an Improved-ACO (IACO) for collaborative offloading in MEC, reducing latency and energy consumption compared to traditional methods. Hussein & Mousa. [17] ACO enhances efficiency by effectively distributing IoT tasks over fog nodes, improving response times and load balancing, thus optimizing overall service quality. Xueli An et al. [18] presents an energy-efficient MEC solution using a two-level ACO and deep deterministic policy gradient (DDPG) framework for task offloading, ensuring reliable task completion and scalability for IoT demands. It prioritizes energy use over profit maximization. Leguizamon et al. [19] study adapts ACO for subset problems, enhancing its suitability for tasks like the Knsack Problem by refining pheromone updating and integrating heuristic information. This approach improves solution quality and stability, extending ACO's applicability to complex combinatorial optimization. Fidanova et al. [20] applies ACO to the Multiple Knapsack Problem, enhancing solution quality through model bias for effective resource-allocation. This approach improves ACO's adaptability to complex subset problems. Khan et al. [21] introduce an Integer Linear Programming (ILP) based offloading algorithm for MEC that allows the selection of execution modes between local execution and offloading task for devices. This approach optimizes performance and low-latency tasks. Guo et al. [22] developed a flexible offloading framework for MEC networks using array signal processing, allowing antennas to execute tasks at nearby computational access points (CAPs) to optimize costs. Their ACO method effectively reduces system expenses while addressing computation offloading challenges in mobile devices. Bao et al. [23] introduce an ACO-based method for efficient computation offloading in mobile cloud computing, enhancing task allocation between devices and cloud resources while reducing latency and energy consumption. This approach effectively optimizes resource utilization in mobile applications. Wang et al. [24] propose a resource-limited MEC system to optimize task scheduling for latency-sensitive user equipment, aiming to minimize energy consumption while maximizing offloaded tasks. They introduce ACO algorithm, Load Balancing ACO (LBA), to effectively address this multi-objective optimization problem.

Limitations Despite significant advancements in the field, current research presents several limitations. Firstly, many studies primarily focus on either minimizing energy consumption or optimizing resource-allocation, often neglecting the financial aspects that are crucial for mobile network operators. Secondly, while algorithmic strategies, including the ACO, NGA, FGA and multi-objective techniques, have evolved, limitations remain in addressing scalability and real time performance in large-scale multi-cell networks. Finally, many proposed solutions are either too complex for practical implementation or fail to consider the dynamics of fluctuating user demands, leading to suboptimal results in real world applications. The contributions of this manuscript include:

- A proposed approach integrates C-RAN with MEC by utilizing a BBU pool, MEC servers and RRHs. This approach is designed to optimize the execution of user-intensive tasks, ultimately aiming to enhance operational efficiency and maximize profit for network operators.
- This study introduces and evaluates four algorithms—NGA, FGA, Modified SJOORA and ACO—designed to maximize profit in network operations. These algorithms are utilized in contexts where decisions about task delegation and coordinated management of processing and wireless communication resources are required. Each algorithm's performance is assessed to determine its effectiveness in maximizing profitability under different network conditions.
- This study presents solutions employing theNGA FGA, Modified SJOORA, ACO to address the integrated optimization of computation offloading strategies and resource management. To improve the computational speed of these algorithms, a machine learning-based regression model is applied to estimate resource-allocation parameters at each iteration within the evolutionary process. This approach enhances execution efficiency while ensuring optimal performance across the optimization tasks.

- This study presents a distributed optimization framework tailored for multi-cellular MEC systems. Emphasizing an offloading methodology, this approach is designed to maximize network operator profitability while addressing key aspects of resource management. Specifically, it provides guidance on optimizing the use of system resources, distributing network capacity effectively, and increasing operational revenue in edge-based mobile computing platforms. This framework aims to provide actionable guidelines for balancing computational demands with resource availability to achieve optimal performance in multicellular MEC networks.
- NGA, FGA, Modified SJOORA and ACO are utilized with more than 20 iterations, selecting the maximum iteration result to compare the performance of the algorithms. Analysis revealed that variations in the number of UEs directly influence which solution method performs best. Based on this finding, a mechanism is proposed where the server dynamically selects the best algorithm according to the current number of UEs, ensuring optimal performance and resource utilization.

Sections This paper is structured as follows: Section 2 discusses the framework design and outlines the optimization problem. Section 3 introduces the NGA, while Section 4 focuses on the FGA. In Section 5, the Modified SJOORA Algorithm is presented, followed by a discussion on the ACO Algorithm in Section 6. The simulations and results are analyzed in Section 7, and finally, Section 8 concludes the paper with a summary of the main findings.

2. Framework Design

A task-focused C-RAN architecture integrated with MEC is proposed. The C-RAN configuration consists of a centralized pool of BBUs and multiple remote RRHs that collectively establish the radio access layer, while the MEC server is integrated into the BBU pool. The system's structural configuration is initially described, after which comprehensive explanations of both the communication system and the computational framework. The section concludes by defining the profit optimization objective for this combined infrastructure.

2.1. *Network Layout*

This research explores a C-RAN architecture enhanced by the integration of MEC. Building on established methodologies [25,26], the BBU pool employs two types of virtual machines (VMs): MEC servers for managing task offloading and traditional virtual BBUs (vBBUs) for handling communication operations. The MEC servers feature a dedicated, limited-capacity storage pool reserved for task data. The focus of this study is on executing computation processes at the network edge and transmitting the corresponding outcomes, while excluding any costs associated with uploading sensitive data. As shown in Figure 1, the C-RAN system design comprises RRHs linked to a centralized the BBU pool via high speed, fiber-optic fronthaul links, collectively defined as the set $I = \{1, 2, 3, \ldots, I\}$. The RRHs are positioned according to a spatial distribution modeled by a Poisson-Point-Process (PPP), and each is configured with a single antenna to streamline the system architecture. Similarly, UEs, each equipped with an individual antenna, are randomly placed throughout the C-RAN service region and represented as $J = \{1, 2, 3, \ldots, J\}$. The offloading strategy for UEs is modeled using a binary matrix $A = \{\phi_{j,i}\}_{J \times I}$, where $j \in J$ and $i \in I$. A matrix element $\phi_{j,i} = 1$ indicates that UE_j can access the MEC server via RRH_i to perform its task, while $\phi_{j,i} = 0$ signifies otherwise [27]. To guarantee that tasks are prepared for execution, it is assumed that user equipment (UE)-generated data is stored in advance on the edge server. A specific task, denoted as S_j and originating from UE_j is characterized as follows:

$$S_j = (F_j, D_j, T_{j,max}), \qquad j = 1, 2, \ldots, J \tag{1}$$

In this context

- F_j specifies the complete computational workload, measured in CPU cycles, that must be processed by the edge server to finalize the task.
- D_j represents the amount of resulting data generated by the task, which is transmitted back to UE_j after computation on the MEC server.
- $T_{j,max}$ specifies the maximum allowable time for delivering the execution result to the mobile terminal UE_j.

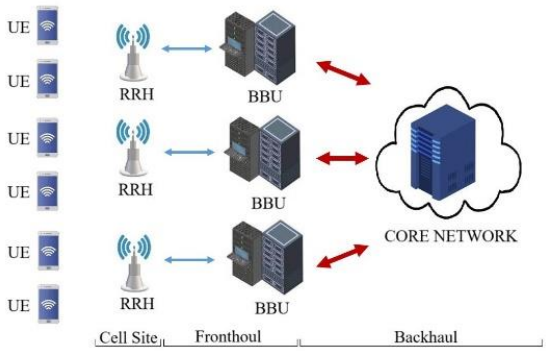


Figure 1. Integrating MEC into Mobile Cloud Systems Framework with (C-RAN) model.

2.2. *Network Communication Architecture*

2.2.1. Channel State Information (CSI) Representation

The CSI refers to data that characterizes the quality of the wireless communication channels and is represented using a matrix that captures the efficiency of spectral usage

$$\varepsilon = \{e_{j,i}\}_{J\times I}, \quad \forall j \in J \text{ and } i \in I, \tag{2}$$

each element $e_{j,i}$ represents the comparative value between the available channel throughput and the corresponding bandwidth assigned, inking a user device UE_j as well as each RRH_i unit. For simulation purposes, the entries of matrix ε are initialized with randomly generated values within the range of 0 to 5, reflecting diverse channel conditions.

2.2.2. Bandwidth Allocation Model

Each RRH has a total available spectrum bandwidth denoted as B Hz. The proportion of bandwidth assigned to a given UEj by an RRHi is represented by the set

$$\rho = \{\rho_{j,i}\}_{J\times I}, \quad \forall j \in J \text{ and } i \in I, \tag{3}$$

with the constraint

$$\sum_{j\in J} \rho_{j,i} \leq 1 \tag{4}$$

to ensure that the total allocated bandwidth does not exceed the available spectrum per RRH.

The relationship between the offloading decision and bandwidth allocation is defined to ensure consistency in the formulation. The binary variable $\varphi_{j,I} \in \{0,1\}$ indicates whether UE_j is associated with RRH_i, while the continuous variable $\rho_{j,i} \in [0,1]$ denotes the fraction of bandwidth assigned from RRH_i to UE_j. To guarantee that bandwidth is allocated only when an active association exists, the following constraint is introduced:

$$\rho_{j,i} \leq \varphi_{j,i}, \quad \forall j \in J, i \in I \tag{5}$$

This additional constraint ensures that no bandwidth is allocated to inactive links, thereby maintaining consistency between the offloading matrix and bandwidth assignment.

2.2.3. Instantaneous Rate Calculation

The achievable instantaneous data rate $(R_{j,i})$ for a UE_j connected to an RRH_i is determined based on the allocated bandwidth fraction ($\rho_{j,i}$) and the corresponding spectrum efficiency ($e_{j,i}$) from the CSI matrix. This relationship characterizes the effective data throughput achievable under the specified bandwidth allocation and channel conditions:

$$R_{j,i} = B \cdot \varphi_{j,i} \cdot \rho_{j,i} \cdot e_{j,i} \tag{5}$$

2.2.4. Fronthaul Capacity Constraints

The total data rate delivered by each RRH to its associated UEs defines a constraint on fronthaul bandwidth capacity. This constraint is formally expressed as:

$$\sum_{j \in J} R_{j,i} \leq \mathrm{CAP}_i, \forall i \in I, \tag{6}$$

where CAP_i denotes the maximum fronthaul capacity of RRH_i, and $R_{j,i}$ denotes the transmission rate from RRH_i to UE_j. This constraint ensures that the combined data rate provided to all UEs served by an RRH does not exceed the RRH's fronthaul capacity.

2.2.5. Transmission Data Time Calculation

The time required for data transmission from RRH_i to UE_j is determined by:

$$T_j^T = \varphi_{j,i} \frac{D_j}{\sum_{i \in I} R_{j,i}} \tag{7}$$

where $\varphi_{j,i}$ denotes the proportion of the data demand D_j handled by RRH_i for UE_j. The denominator, $\sum_{i \in I} \boldsymbol{R_{j,i}}$, represents the total transmission rate obtained by summing the contributions from all RRHs providing offloading support to UE_j.

2.3. *Computation Model*

2.3.1. Task Offloading and Resource-Allocation in MEC Server

After defining an offloading plan and identifying the user devices suitable for workload delegation, the edge computing server distributes processing resources to each device in terms of computational cycles per second. If a specific device UEj is permitted to execute the workload remotely the MEC server processes the request by locating the required pre-stored information and initiating task execution. The total computational capacity of the MEC server is denoted by F, representing the server's processing speed in cycles per second.

Within this framework, the computational resources are shared among all devices (UEs) permitted for offloading. The fraction of computing resources allocated to each UEj is represented by $c_j \in [0, 1]$, with the constraint

$$\sum_{j \in J} \boldsymbol{c_j} \leq \mathbf{1}, \tag{8}$$

to ensure efficient resource distribution. The set of allocated resource fractions is denoted by $c = \{c_j\} \ \forall j \in J$.

2.3.2. Execution Duration Calculation

The duration required to execute the task Sj of UEj on the MEC server is given by:

$$T_j^{ex} = \varphi_{j,i} \frac{F_j}{c_j F} \tag{9}$$

where:

- F_j : Total CPU cycles necessary to complete task S_j.
- $c_j F$: Computing resources, measured in cycles per second, allocated to UE_j by the edge computing server.

2.3.3. Total Offloading Time

The total time cost for UE_j to complete the task offloading process, combining transmission and execution times, is expressed as:

$$T_j = T_j^T + T_j^{ex} \tag{10}$$

2.3.4. Resource-Allocation Constraint for Offloading Feasibility

To ensure offloading is beneficial, the computing resources for a user device located within the MEC server must surpass the UE's local computing capacity ($\boldsymbol{F_j^{local}}$). This condition is satisfied if:

$$c_j F \leq \sum_{i \in \mathrm{I}} \varphi_{j,i} F_j^{local} \tag{11}$$

2.3.5. Profit Modeling and Objective Formulation

The study aims to optimize the network operator's profit by formulating a profit objective function that balances revenue and cost. The revenue generation aspect focuses on task processing and data caching within a MEC server, where the network operator charges UEs based on computational and data transmission services.
Revenue Function: The total revenue for task S_j, when offloaded by UE$_j$, is given by:

$$\Omega_j = \sum_{i \in I} \varphi_{j,i} (Y_f F_j + Y_t D_j + Q_j) \tag{12}$$

where $\varphi_{j,i}$ is a binary variable indicating whether UE$_j$ receives offloading support from RRH$_i$ (i.e., $\sum_{i \in I} \varphi_{j,i} \neq 0$).

Cost Function: The cost function includes the following components:

- **Spectrum Cost**: Based on the assigned bandwidth for UEj priced at H_b per hertz of spectrum.
- **Computational Resource Cost:** Determined by the processing power designated for task S_j, charged at H_c per unit of CPU processing rate.

The total cost for offloading task S j is given by:

$$cost_j = \sum_{j \in J} \sum_{i \in I} \varphi_{j,i} \left(k c_{mj} F H_c + \omega \rho_{j,i} B H_b \right) \tag{13}$$

where k reflects variability in pricing between bandwidth and processing resources, and ω indicates the balancing weight used to account for resource availability.
Profit Function: The profit for an offloading-enabled UEj is:

$$U_j = \Omega_j - cost_j \tag{14}$$

The total profit function for all user devices with offloading capability is represented as:

$$U = \sum\nolimits_{j \in J} \sum\nolimits_{j \in I} (\Omega_j - cost_j) \tag{15}$$

2.4. *Profit Maximization Problem and Constraints*

To maximize profit, the objective function is defined as:

$$\max_{\varphi_{j,i}, \rho_{j,i}, c_j} \sum_{j \in J} \sum_{i \in I} (\Omega_j - cost_j) \tag{16.a}$$

subject to the following constraints:

- **C1:** Total allocated bandwidth at each RRH does not exceed its available capacity:

$$\sum_{j \in J} \varphi_{j,i} \rho_{j,i} \leq 1 , \quad \forall i \in I \tag{16.b}$$

- **C2:** Limits on computational resources within the MEC server:

$$\sum_{j \in J} \sum_{i \in I} \varphi_{j,i} c_j \leq 1 \ \forall i \in I , \forall j \in J \tag{16.c}$$

- **C3:** Quality of Service (QoS) constraint ensuring total execution time meets latency requirements:

$$T_j^T + T_j^{ex} \leq T_{j,max} , \forall j \in J \tag{16.d}$$

- **C4:** Computational resources allocated to a UE must exceed its local capacity:

$$c_j F \geq \varphi_{j,i} F_j^{local} , \forall j \in J , \forall i \in I \tag{16.e}$$

- **C5:** The cumulative data rate of all UEs served by an identical RRH must not surpass the data transfer limit of the RRH's fronthaul connection.

$$\sum_{j \in J} R_{j,i} \leq CAP_i \ , \forall \ i \in I \tag{16.f}$$

- **C6:** $\varphi_{j,i}$ is a binary decision variable**:**

$$\varphi_{j,i} \in \{0,1\} \ , \forall \ j \in J, \forall \ i \in I \tag{16.g}$$

3. Applying Normal Genetic Algorithm for Efficient Task Offloading

NGAs are optimization techniques inspired by natural selection, designed to minimize objective functions by refining candidate solutions across generations. Starting with a population of potential solutions represented as chromosomes, NGAs evaluate each based on a fitness measure tied to the objective function. High-performing candidates (those that achieve lower values) are selected to reproduce, with genetic operators like crossover and mutation applied to generate offspring. Crossover combines traits from parent solutions, while mutation introduces random changes, promoting diversity and helping the algorithm avoid local minima [28].

This iterative process of selection, crossover, and mutation guides the population toward solutions that minimize the objective function. Termination occurs when the algorithm achieves a satisfactory solution or reaches a set number of generations [29,30]. NGAs effectively balance exploration and refinement within the search space, making them suitable for both constrained and unconstrained minimization problems and often yielding competitive results compared to other optimization methods.

4. Applying Fast Genetic Algorithm for Efficient Task Offloading

The conventional NGA often requires significant computational resources, especially when applied to optimization problems that include constraints, such as maximizing a profit function. Repeatedly computing this profit function across a large initial population can result in inefficient resource utilization. To mitigate this, a smaller initial population is often chosen, although this conflicts with the fundamental principle of NGAs, which generally depend on a large and diverse initial population for effective evolutionary progression. In contrast, a fast optimization approach integrates the problem's constraints directly into the objective function as penalty terms [31], chosen for their simplicity and efficiency. Penalty parameters are adjusted based on the objective function's dimensionality to discourage infeasible solutions strongly [32], allowing the algorithm to start with a broader range of solutions and converge to an optimal population more rapidly than a traditional GA.

In the FGA, constraints are handled directly within the profit calculation, allowing for a larger initial population and reducing computational time. The highest-ranking five candidates from the final population are evaluated using MATLAB's optimization solver, selecting the individual with the highest profit as the final solution. To approximate profit in FGA, a decision tree regression model in MATLAB is used to predict wireless spectrum allocation and computational resource distribution, ensuring accurate resource estimates. This model is built from an optimization-specific dataset generated by solving the problem repeatedly with random offloading strategies, filtering out infeasible results [33] , [34]. Additionally, a penalty term was added to the profit function to address constraint violations, ensuring the required constraints (C1, C2, C6) are met. The main difference between NGA and FGA lies in profit calculation: while the NGA uses a smaller population and optimization solvers, the FGA employs a heuristic approach, incorporating constraints into the objective function to effectively handle a larger, more diverse population.

In the simulation framework, the population size of the NGA was restricted to 15 individuals because the evaluation of each candidate requires repeated execution of the optimization solver, which is computationally expensive. Using larger populations under this setting would result in impractical runtimes and high resource consumption. By contrast, the FGA incorporates constraint-handling directly into its objective function through penalty terms, thereby avoiding repeated solver calls and substantially reducing computational complexity. This efficiency enables FGA to employ much larger populations—up to 2000 individuals—allowing greater search diversity and faster convergence while remaining computationally feasible based on the work by Singh et al. [32].

5. Applying Modified SJOORA Algorithm for Efficient Task Offloading

The "Spectrum Efficiency (SE)-based Joint Optimization for Offloading and Resource-Allocation (SJOORA) scheme" is developed to enhance the performance of task delegation and resource distribution within wireless communication networks. In these systems, task delegation refers to offloading computationally demanding processes from user devices to distant servers or cloud-based platforms. This offloading reduces the computational load on mobile devices, leading to more efficient resource usage. However, decisions regarding offloading must account for various factors, including network conditions, energy consumption, and the computational capacities of both mobile devices and remote resources [35].

A modified version of the SJOORA framework was introduced in [35], where the authors proposed a spectral efficiency-based joint optimization of radio and computing resource allocation (SJOORA) to determine the offloading strategy defined in (16.g). In this work, we extend that approach by employing a customized NGA to approximate the optimal offloading strategy and compare its performance against the SJOORA scheme. The NGA requires evaluating random populations of candidate solutions, and although the objective function in (16.g) is computationally intensive, the algorithm enables efficient approximation of near-optimal solutions. The Modified SJOORA scheme focuses on improving spectral efficiency by jointly optimizing task migration, computational resource allocation, and network resource distribution. By addressing offloading and resource management simultaneously, it enhances data transmission capabilities and improves overall system performance.

6. Applying Ant Colony Algorithm for Efficient Task Offloading

The ACO algorithm is a nature-inspired metaheuristic technique that mimics the foraging behavior of ants to solve complex optimization problems. It has been effectively applied in 5G C-RAN environments to address challenges such as resource-allocation and task scheduling [36]. The algorithm's key objective in this context is to maximize network profit and minimize execution time by optimizing how user demands are distributed across BBUs and RRHs. This balance is critical for ensuring high efficiency and low latency in 5G communication systems [24].

In C-RAN, the ACO algorithm models the resource-allocation problem as a multi-objective optimization task. Each ant represents a potential solution, mapping user demands to available network resources while minimizing processing delays. The probability Pi j of an ant moving from node i to node j is defined as [37,38]:

$$P_{ij} = \frac{\tau_{ij}^{\alpha} . \eta_{ij}^{\beta}}{\sum_{k \epsilon N} \tau_{ik}^{\alpha} . \eta_{ik}^{\beta}} \quad (17)$$

where:

- P_{ij}: The probability of transitioning from node i to node j.
- τ_{ij}: The pheromone level on the path from i to j, representing the desirability of that path based on past solutions.
- η_{ij}: The heuristic value of the path, which can depend on factors like the computational capacity of BBUs or the latency of network links.
- α: A control parameter that determines the influence of the pheromone trail.
- β: A control parameter that determines the influence of the heuristic information.
- N: The set of possible next nodes.

In the context of our resource allocation problem, the ACO transition rule (Eq. 17) is adapted as follows. Each node corresponds to a potential offloading decision, specifically the assignment of a task generated by UE_j to an RRH_i with a feasible share of bandwidth and MEC computing resources. A complete ant tour therefore represents a full offloading strategy, where all UEs are sequentially associated with RRHs and their resource shares are determined. The pheromone value τ_{ji} quantifies the historical quality of assigning UE_j to RRH_i, while the heuristic factor η_{ji} captures the instantaneous channel condition, available bandwidth, and expected latency cost for that assignment. In this way, the probabilistic transition rule governs how ants iteratively construct feasible offloading solutions, linking the standard ACO mechanism directly to the MEC–C-RAN allocation problem.

The algorithm relies on pheromone updates to refine solutions. After ants complete their tours, pheromone trails are updated using the formula:

$$\Delta\tau_{ij} = \frac{\mathrm{Q}}{\mathrm{Fitness}} \tag{18}$$

where:

- $\Delta\tau_{ij}$: The amount of pheromone deposited on the path from i to j.
- Q: A constant scaling factor that controls the magnitude of the pheromone deposit.
- Fitness: The quality of the solution, combining profit and execution time.

The choice of Q depends on the scale of the problem and the desired balance between exploration and exploitation. In a typical C-RAN problem, Q should be chosen based on the range of fitness values in the system. For instance, if the fitness values are expected to range from 1000 to 10,000, Q might be set in a range like 1000 to 5000. This scaling ensures that pheromone updates are significant enough to guide ants toward high-quality solutions without overwhelming the search process. Q can also be dynamically adjusted during the execution of the algorithm to reflect the algorithm's progress—starting with a larger Q for exploration in the early stages and gradually decreasing it to focus on exploitation of good solutions as the algorithm converges.
The fitness function used to evaluate solutions is:

$$Fitness = w_1 \cdot Profit - w_2 \cdot Execution\,Time \tag{19}$$

where:

- w_1: Weight representing the importance of maximizing profit.
- w_2: Weight representing the importance of minimizing execution time.
- Profit: The revenue or benefit gained from the allocated resources.
- Execution Time: The total time taken for task completion.

Weights w_1 and w_2 are determined based on network priorities. For instance, higher w_1 emphasizes profit, while higher w_2 prioritizes reducing latency. These weights are normalized to ensure consistency and are often tuned through experimental or automated methods like grid search [38].
The scalability and adaptability of ACO make it ideal for dynamic environments like C-RAN. It can efficiently handle large-scale optimization problems and adapt to changes in traffic patterns or resource availability. However, challenges such as parameter tuning (α, β, Q, and pheromone evaporation rate) must be addressed to balance exploration and exploitation. Future enhancements, such as integrating ACO with machine learning or hybrid optimization methods, can help meet real-time constraints and further improve performance. By leveraging ACO, operators can achieve efficient resource utilization, reduce energy consumption, and meet the stringent latency requirements of 5G networks.
For clarity, the pseudocode (Algorithm 1) and the flowchart (Figure 2) of the proposed ACO scheme are presented at the end of this section. The pseudocode summarizes the main stages—initialization of pheromones and heuristics, construction of candidate offloading solutions, fitness evaluation based on profit and latency, and pheromone updates. The accompanying flowchart provides a visual overview of this iterative process, showing how constraints such as bandwidth, latency, and MEC capacity are enforced. Together, they offer a concise yet comprehensive representation of the ACO-based offloading strategy.

Algorithm 1 Ant Colony Optimization for MEC Resource Allocation

Input: Number of UEs, RRHs, ACO parameters (α, β, ρ), system parameters
Output: Optimal allocation of UEs to RRHs, maximum profit, performance metrics
Initialize pheromone levels τ_{ij} for all UE-RRH pairs
for each UE count M **do**
    Generate random UE and RRH distributions
    Calculate energy efficiency matrix for UE-RRH pairs
    for each iteration $k = 1$ to k_{max} **do**
        Initialize ant solutions
        for each ant i **do**
            for each UE j **do**
                Calculate selection probability P_{ij} based on pheromone levels and heuristic values
                Assign UE j to an RRH based on P_{ij}
            end for
            Evaluate the profit and system performance for the solution
        end for
        Update pheromone levels based on solution quality
    end for
    Store the best solution and its metrics
end for
Return the best allocation and performance metrics

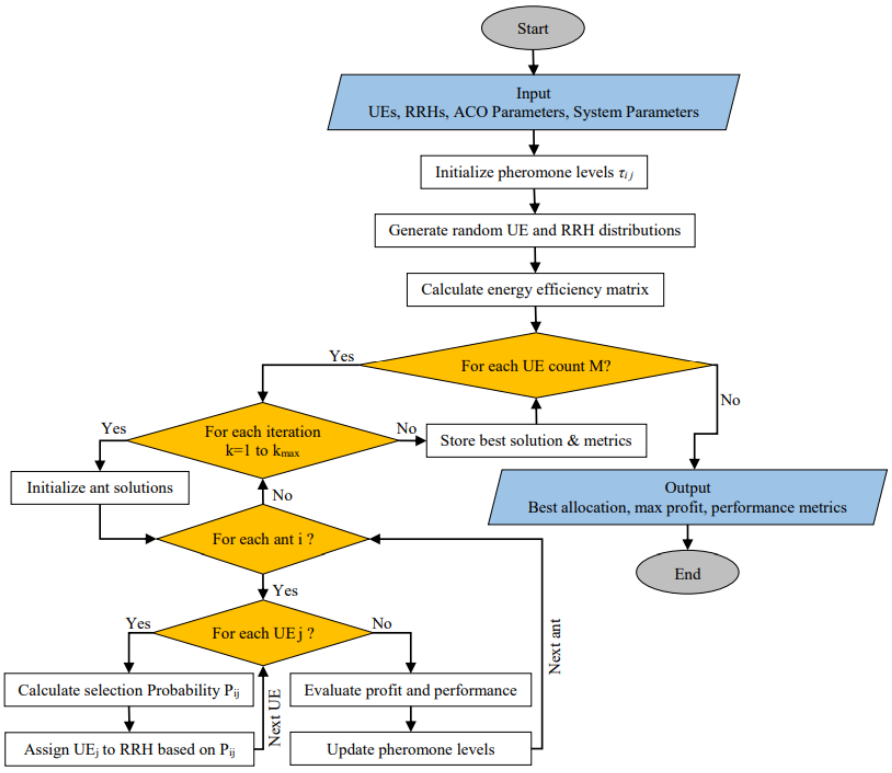


Figure 2. Flowchart of the ACO-Based Task Offloading Strategy in MEC–C-RAN Systems

7. Simulation and Results

Simulation experiments were conducted using MATLAB, with parameter values drawn from the literature [32] to simulate a wireless system configuration. In the simulation setup, UEs were distributed randomly across the coverage area, with the number of UEs (denoted by J) varying between 30 to 110 to fulfill the simulation requirements. Ten RRHs, represented as $I = 10$, were used, each with a fronthaul link capacity of 50 Mbps.
Tasks were created using randomized parameters, with latency requirements ranging from 360 to 900 ms to ensure precise control for efficient task processing. Output data size, denoted as D_j, varied between 50 and 200

kbytes, highlighting the need for scalable data management. The computational demand for each task, denoted as F_j, depended on its data size and was expressed through the following relation:

$$F_j = 10^3 \times D_j \tag{20}$$

Additionally, A matrix of dimensions J by I containing values randomly selected within the range of 0 to 5 was used to simulate varying spectrum efficiency. Additional parameters crucial for effective task management are compiled and displayed in Table 1 [35], providing detailed insights into task optimization strategies.
For wireless access, bandwidth B for the communication channel was set to 10 MHz, while each RRH transmitted at a power of 30 dBm. Additive White Gaussian Noise (AWGN) was mode led with a noise power level of $\sigma^2 = -174$ dBm/Hz. Path loss was modeled based on the formula:

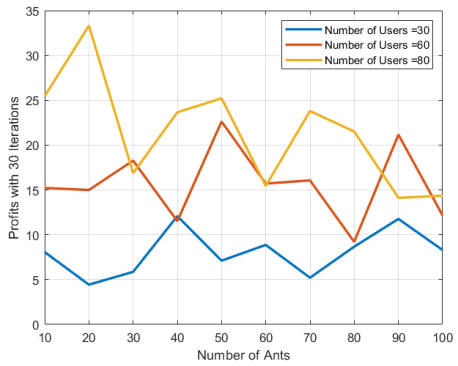
$$\text{Path Loss (dB)} = 37.6 \cdot \log(\text{dist}) + 148.1 \tag{21}$$

which accurately represented signal attenuation over distance in the simulated environment [39]. These parameter settings collectively provided a robust framework for assessing the performance and efficiency of task processing and resource-allocation in a wireless network scenario.

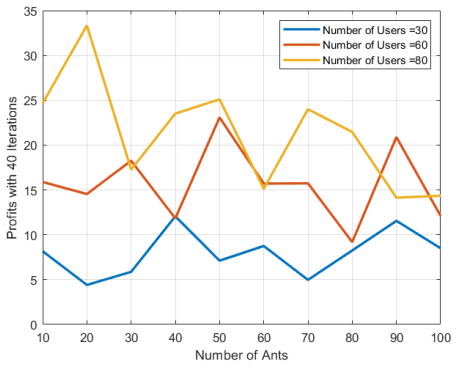
Table 1. Additional Simulation Parameters

Parameter	Value
Storage cost Q_j	0.1 $
Bandwidth cost impact factor k	0.5
Computation cost impact factor ω	10
Charge unit price for computation Y_f	0.03 $ / Mega cycle
Charge unit price for transmission Y_t	0.3 $ / Mbit
Bandwidth unit cost H_b	0.5 $ / MHz
Computation unit cost H_c	0.005 $ / Mega cycle/s
MEC server capacity F	100 GHz
Local computational capability F_j^{Local}	0.7 GHz
Number of individuals (NIND)	15 (Normal), 2000 (Fast)
Generation gap (GGAP)	0.8
Mutation rate (MUTR)	0.05
Maximum generations (MAXGEN)	3

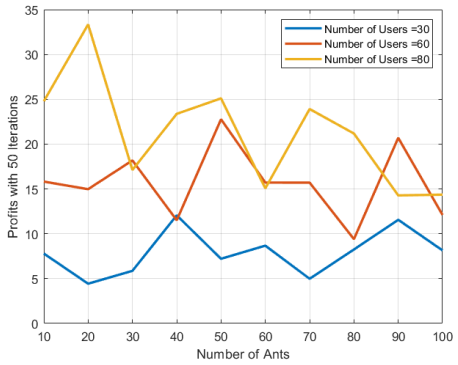
7.1. *Comparison of Profits Across Iteration Counts (30, 40, and 50) for Different Numbers of Users and Ants*



(a) Number of Ants 30



(**b**) Number of Ants 40



(**c**) Number of Ants 50

Figure 3. Overview of profits for different numbers of ants.

This study shown in figure 3 examines the impact of iteration counts (30, 40, and 50) on profit optimization for varying numbers of users (30, 60, and 80) and ants (10 to 100). The results indicate that increasing the number of iterations has a negligible effect on profit optimization. Across all iteration counts, 80 users consistently achieve the highest profits, peaking at approximately 20 ants, with maximum profits remaining nearly constant at 33.34 for both 40 and 50 iterations. Similarly, trends for 60 and 30 users show minimal variation in profits with increasing iterations. This suggests that while iteration count slightly stabilizes trends, it does not significantly enhance optimization or profitability, emphasizing that user count and the number of ants are the primary factors influencing performance.

7.2. *Analysis of Execution Time in Relation to the Number of Ants and Users*

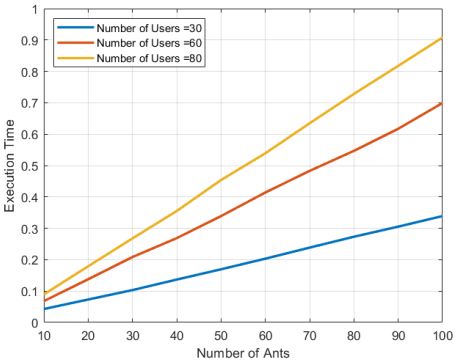


Figure 4. Execution Time vs Number of Ants for Different Numbers of Users

Figure 4 displays the execution time as a function of the number of ants for three different user counts (30, 60, and 80) based on the new data. Execution time increases steadily as the number of ants grows across all user configurations. However, a noticeable trend is that 30 users experience the shortest execution times, followed by 60 users, with 80 users having the longest execution times for each corresponding number of ants. For example, at 100 ants, the execution time is highest for 80 users (0.907 seconds) and lowest for 30 users (0.339 seconds). This pattern suggests that the efficiency of the system decreases with the number of users, likely due to the increased computational demand as the system scales.

7.3. *Analysis of Franthaul Performance Across User Groups and Ant Configurations*

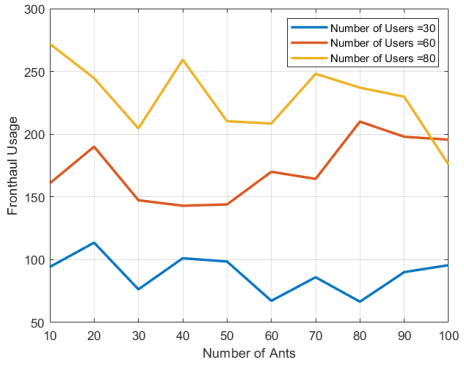


Figure 5. Franthaul vs. Number of Ants for Different User Densities

Figure 5 illustrates the analysis of fronthaul performance demonstrates clear trends and offers insights into future expectations. For the 30-user group, the franthaul varies a lot, reaching a peak of 113.57 Mbps with 20 ants but dropping to 66.55 Mbps at 80 ants. This suggests that resource-allocation is inconsistent for this group. In the 60-user group, the performance is more stable, with franthaul ranging from 142.98 Mbps to 209.97 Mbps, indicating more efficient resource use. The 80-user group starts with the highest franthaul at 271.91 Mbps with 10 ants, but the performance drops significantly as the number of ants increases, falling to 175.73 Mbps at 100 ants. This decline could be due to resource saturation or inefficiencies in handling higher demands. As the number of users increases, the system will face greater pressure, and franthaul demands will grow. To handle this, it's essential to optimize the number of ants to ensure consistent performance and avoid bottlenecks, especially as user density rises.

7.4. *Analysis of MEC Computation Times Across User Groups and Ant Configurations*

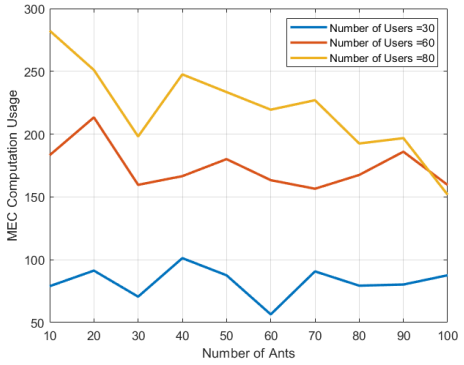


Figure 6. MEC Computation vs. Number of Ants for Different User Densities

Figure 6 illustrates the analysis of MEC computation times across varying numbers of ants for different user groups is highlighted, showcasing distinct trends in resource utilization. For the 30-user group, computation times fluctuate significantly, peaking at 101.27 ms with 40 ants and dropping to a minimum of 56.50 ms with 60 ants, indicating inconsistent efficiency. The 60-user group demonstrates more stable performance, with computation times ranging from 156.53 ms (70 ants) to 213.35 ms (20 ants), reflecting consistent but suboptimal resource-allocation in certain cases. The 80-user group exhibits the highest computation times, starting at 282.36 ms with 10 ants and steadily declining to 151.75 ms with 100 ants, suggesting improved efficiency at higher ant numbers but diminishing returns at the extremes. These results indicate that the number of ants significantly influences MEC computation times, and optimizing this parameter is critical to achieving efficient performance, particularly as user density increases.

7.5. *Analysis of Power Consumption Across User Groups and Ant Configurations*

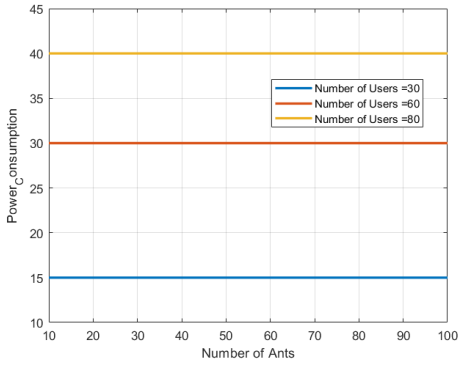


Figure 7. Power Consumption vs. Number of Ants for Different User Densities

Figure 7 is presented, where the analysis of power consumption across varying numbers of ants for different user groups is revealed, showing consistent trends. For all user groups (30, 60, and 80 users), power consumption remains remarkably stable across the range of ants, with minimal fluctuations. In the 30-user group, power consumption consistently hovers around 15 W, while in the 60-user group, it remains near 30 W. Similarly, for the 80-user group, power consumption stabilizes around 40 W. This stability indicates that power consumption is primarily determined by user density rather than the number of ants, with slight variations being negligible in practical scenarios. The data suggests that the power consumption scales linearly with the number of users, remaining unaffected by the configuration of ants within the tested range. These findings underscore the efficiency of the system in maintaining predictable energy usage regardless of ant allocation, which is critical for energy optimization and system reliability.

7.6. *Analysis of Power consumption vs. Number of Users*

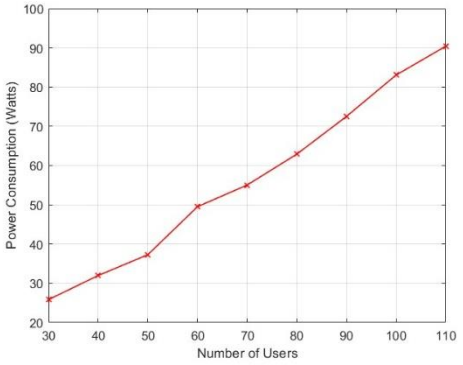


Figure 8. Power Consumption as a Function of the Number of Users

Figure 8 illustrates the relationship between the number of users and power consumption is depicted in the provided data and graph, showing a steady increase as the user count rises from 30 to 110. Power consumption starts at 25.843 Watts for 30 users and grows nearly linearly, reaching 90.392 Watts for 110 users. Notable increments are observed at higher user counts, such as a 12.23-Watt increase between 50 and 60 users, suggesting potential inefficiencies or system thresholds under heavier loads. This trend highlights the proportional scaling of power consumption with user activity, offering insights into system performance and energy demand management.

7.7. *Analysis of Fronthaul Link Capacity vs. Number of Users*

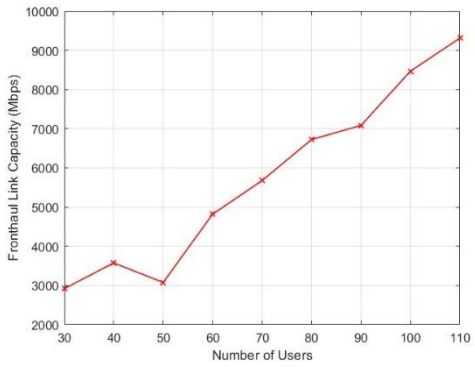


Figure 9. Fronthaul Link Capacity vs. Number of Users

Figure 9 illustrates the relationship between the fronthaul link capacity (measured in Mbps) and the number of user's equipment (denoted as UE) within a network. The trend suggests a generally increasing capacity requirement as the number of users rises. Starting from 30 users, the fronthaul link capacity begins around 2919Mbps, exhibiting fluctuations but following an upward trend overall. As the user count increases, the demand on the fronthaul link capacity escalates, particularly notable at higher user counts (e.g., between 90 and 110 users), where the capacity surpasses 8000 Mbps. This increase reflects the additional data throughput needed to accommodate more users within the network, implying a correlation between user growth and required link capacity. The graph indicates some nonlinear growth, with certain user intervals showing more significant capacity jumps. This pattern may imply points at which network resources experience greater strain or congestion, likely due to varying data requirements of different applications or user behaviors.

7.8. *Computational Capacity Requirements for MEC Servers Based on User Load*

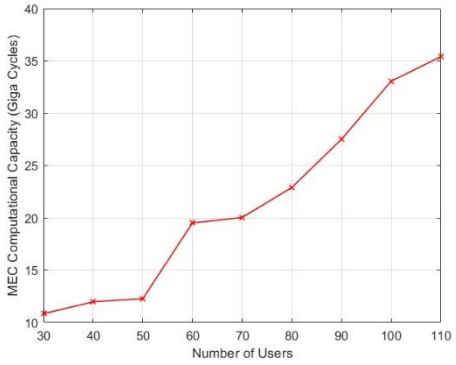


Figure 10. MEC Server Computational Capacity vs. Number of Users

Figure 10 displays the relationship between the number of users equipment UEs and the re- quired computational capacity of a MEC server, measured in Giga Cycles. As the number of users increases, there is a clear upward trend in the computational capacity requirement. The MEC computational capacity starts at around 10 Giga Cycles for 30 users, then exhibits a progressive increase, particularly sharp between 40 and 50 users. This pattern suggests that as more users connect to the MEC server, the processing demand grows, requiring greater computational resources to handle the increased data load. The growth, however, is not strictly linear, with some intervals showing more rapid rises in capacity demand than others. Notable jumps, such as between 50 and 60 users, could indicate points where the MEC server experiences greater stress due to higher workloads. This trend highlights the importance of scalable resource-allocation in MEC systems, where the computational capacity must be managed to meet varying user demands while ensuring optimal performance.

7.9. *Execution Time Analysis of Optimization Algorithms*

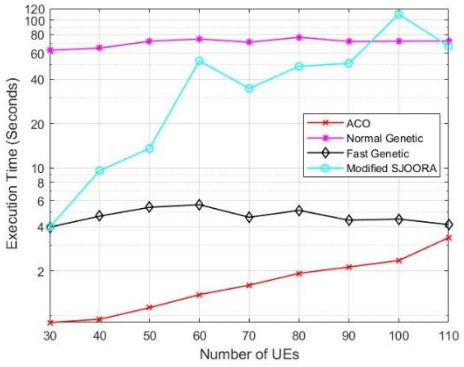


Figure 11. Comparison of Execution Times of the Algorithms

The execution times of the algorithms in this study depend significantly on the hardware used. MATLAB experiments were run on a PC with an 11th Gen Intel Core i7-11800H processor at 2.3 GHz and 16 GB RAM. This setup provides necessary context for interpreting the reported performance results. The graph presented in Figure 11 illustrates the execution times of four algorithms—ACO, NGA, FGA, and Modified SJOORA—over an increasing number of UEs. The y-axis represents execution time in seconds, and the x-axis denotes the number of UEs. A logarithmic scale has been applied to the y-axis to improve clarity, especially given the significant variations in execution times among the algorithms.

Table 2. Comparison of Execution Times of Algorithms Across Different Numbers of UEs

Number of UEs	ACO (seconds)	NGA (seconds)	FGA (seconds)	Modified SJOORA (seconds)
30	0.895733	62.604	3.96293	3.96293
40	0.941266	64.7591	4.70752	9.5842
50	1.12774	72.0355	5.40063	13.5278
60	1.38255	74.7224	5.63	53.168
70	1.60174	70.9226	4.62452	34.493
80	1.92797	76.7703	5.15016	48.6929
90	2.12806	71.7821	4.41826	51.0491
100	2.35468	72.0974	4.49485	109.944
110	3.36302	72.2174	4.12802	66.7694

The NGA shows relatively constant execution times, averaging around 70 seconds regardless of the number of UEs. This behavior suggests that the performance of NGA is unaffected by UE quantity. However, the Normal GA takes longer than the FGA due to the computational effort required for fitness evaluation through an optimization solver at each generation. In contrast, the FGA maintains nearly constant execution times of approximately 4 seconds. The FGA's reduced computational time is attributed to its approximate profit calculation for offloading decisions, which requires fewer resources compared to the NGA. The Modified SJOORA algorithm displays increased execution times as the number of UEs rises. While it initially matches the performance of the FGA, its execution time escalates as the UE count grows, eventually exceeding 60 seconds with higher UE counts. This increase in execution time aligns with the linear time complexity of the original SJOORA algorithm, which becomes more apparent as UEs increase. Finally, the ACO algorithm shows a gradual increase in execution time, yet it remains faster than the Modified SJOORA and NGA, even at higher UE counts. The ACO's moderate scaling behavior makes it suitable for applications where execution speed is crucial but may not match the speed of the FGA for smaller workloads. The logarithmic scale in the plot was used to better

visualize the differences in execution times between the algorithms, particularly since the Normal GA and Modified SJOORA exhibit much higher execution times than ACO and FGA. Without the log scale, the differences between these algorithms would be less discernible, making it challenging to compare performance across the entire UE range.

7.10. *Comparison of Maximum Offloading Strategies*

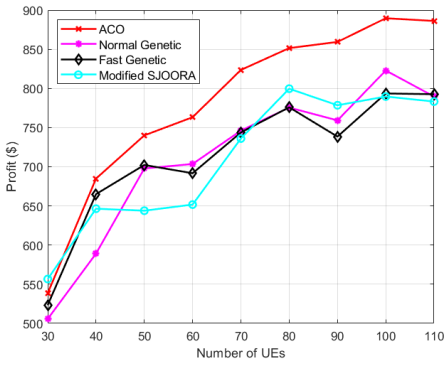


Figure 12. Comparison of Maximum Offloading Strategies

Table 3. Profit Comparison of Algorithms Across Different Numbers of UEs

Number of UEs	ACO ($)	NGA ($)	FGA ($)	Modified SJOORA ($)
30	538.578	505.706	523.48	556.645
40	684.888	589.124	665.026	646.417
50	740.046	697.778	702.255	643.888
60	763.25	703.52	691.71	651.71
70	823.62	746.109	743.788	736.049
80	851.5	775.586	775.904	799.643
90	859.39	759.2	738.495	778.53
100	889.63	822.886	793.573	789.659
110	886.18	789.681	792.624	783.232

Figure 12 illustrates the relationship between the number of UEs and the profit (in dollars) across four algorithms: NGA, FGA, Modified SJOORA, and ACO. The x-axis represents the number of UEs, ranging from 30 to 110, while the y-axis shows the corresponding profit values. As the number of UEs increases, the ACO algorithm demonstrates a significantly higher profit than the other three algorithms, especially at higher UE counts. This trend indicates that ACO's performance scales effectively with an increasing number of UEs, showing a notable rise in profitability, reaching around $886.18 for 110 UEs. In contrast, both NGA and FGA show more gradual profit increases as the UE count rises. NGA's profit grows from around $506 at 30 UEs to about $790 at 110 UEs, while FGA's profit progresses from roughly $523 to $793 across the same range. Although FGA tends to perform slightly better than NGA at lower UE counts, both algorithms exhibit limited overall growth, with profits stabilizing under $800. This trend suggests a less substantial response to increasing UE numbers compared to ACO, indicating potential limitations in their scalability. The Modified SJOORA algorithm displays a pattern similar to that of NGA and FGA, showing limited profit growth as UE count increases. Profit for Modified SJOORA starts at approximately $556 for 30 UEs and reaches around $783 for 110 UEs. This modest growth and overall stability in profit values suggest that Modified SJOORA may lack the robust scalability exhibited by ACO, as it remains well below ACO's profit levels, even at higher UE densities. This disparity suggests that ACO may be better suited for scenarios with high UE densities, potentially due to its optimization mechanism

that allows it to handle complex decision-making more efficiently than the other algorithms. The other algorithms appear to have reached a saturation point where increasing the number of UEs does not yield substantial profit improvements. This could be attributed to limitations in their optimization strategies, which might not be as adaptive to scaling as ACO.

7.11. *Execution Time and Profit Analysis*

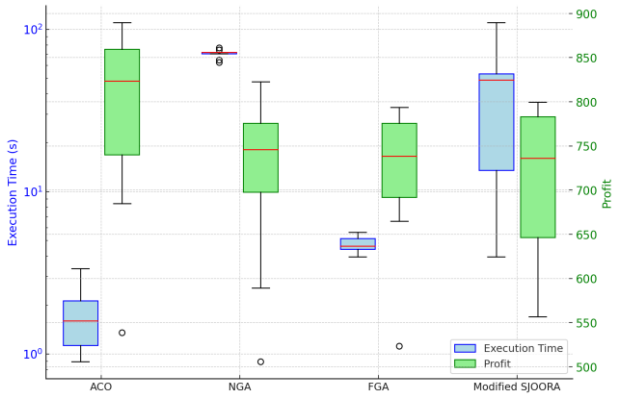


Figure 13. Execution time and profit for the evaluated algorithms.

Figure 13 shows box plots of execution time (blue, left axis, logarithmic scale) and profit (green, right axis, linear scale) for ACO, NGA, FGA, and Modified SJOORA across user counts ranging from 30 to 110. Each box represents the interquartile range, the horizontal red line indicates the median, and the whiskers extend to the minimum and maximum values. The results demonstrate that ACO consistently achieves higher profit while maintaining substantially lower execution times than the other algorithms, confirming its efficiency and scalability in high-density network scenarios.

8. Discussion

The evaluation demonstrates that algorithm choice in MEC–C-RAN resource allocation involves inherent trade-offs between execution time, profit, scalability, and energy efficiency. Among the tested approaches, Ant Colony Optimization (ACO) achieved the most balanced outcomes, consistently maximizing profit while maintaining moderate execution times, making it particularly effective in dense network environments. FGA offered the lowest runtimes but lacked scalability at higher user densities, NGA produced stable but less competitive results, and Modified SJOORA showed acceptable performance only in small- to mid-scale deployments. Analysis of power consumption and fronthaul capacity further indicated that overall system load, rather than algorithmic parameters, is the dominant factor driving computational demand and energy use.

When benchmarked against Particle Swarm Optimization (PSO) variants, namely Sticky Binary PSO (SBPSO) and Dynamic Sticky Binary PSO (DSBPSO), ACO also demonstrated clear superiority. SBPSO employs a stickiness mechanism to reduce oscillations, and DSBPSO adaptively updates parameters to improve exploration; however, both approaches exhibited scalability and convergence limitations under heavy user demand. By contrast, ACO consistently outperformed both methods across all user equipment (UE) levels, with the performance gap becoming more pronounced beyond 60 UEs.

Table 4. Profit comparison of ACO with recent research across different UE numbers

Number of UEs	SBPSO Profit ($) [40]	DSBPSO Profit ($) [40]	ACO Profit ($)
30	445	450	550
40	530	540	680
50	610	620	740
60	640	650	765
70	700	710	810
80	740	750	840
90	780	790	855
100	810	820	870
110	835	850	890

As summarized in Table 4, the results from Singh and Kim [40] are included for comparison purposes only. They highlight ACO's robustness in high-density networks, surpassing DSBPSO by $30–$60 and yielding even larger margins over SBPSO in the 70–110 UE range. This advantage is driven by ACO's pheromone-based reinforcement mechanism, which balances exploration and exploitation, prevents stagnation, and maintains diverse search paths. Unlike SBPSO, which often converges prematurely, and DSBPSO, which only partially alleviates this issue, ACO ensures effective offloading and resource allocation even as system complexity rises. Importantly, these benefits are not achieved at the expense of runtime: although ACO is slightly slower than lightweight approximations such as FGA, it remains faster than NGA and Modified SJOORA while producing substantially higher profits.

Several methodological innovations strengthen these results. First, our study emphasizes profit-centric optimization, addressing a gap in prior MEC–C-RAN research where latency or energy minimization dominated. By embedding a detailed economic model (Equations 8–11), profit is directly linked to operator revenue and resource costs. Second, a novel fitness function (Equation 15) explicitly balances profit maximization and execution time, allowing tunable trade-offs based on operator priorities. Third, a dynamic pheromone update strategy is proposed, where parameter Q is scaled according to fitness ranges and iteration progress, improving convergence quality compared with fixed settings. Fourth, the work benchmarks ACO against diverse algorithmic families (NGA, FGA, Modified SJOORA) under identical conditions, confirming ACO's superior scalability and profitability, particularly as UE counts exceed 60. Finally, the system-aware heuristic design (Equation 13) leverages C-RAN-specific parameters—including channel state information, computation capacity, and fronthaul limits—to guide the search process toward feasible and high-performing solutions.

The scalability of ACO was further validated through extensive simulation experiments in which the system size was varied from 30 to 110 UEs, covering both small-cell and dense urban deployment scenarios. The results confirm that ACO scales effectively in terms of both profit and execution time. Profitability continued to rise with increasing user density, clearly outperforming other algorithms at higher scales, reaching about $889 for 100 UEs compared with approximately $793 for FGA and $823 for NGA. At the same time, execution time increased moderately with the number of UEs, from around 0.9 seconds for 30 UEs to about 3.4 seconds for 110 UEs. This manageable growth in runtime, while maintaining superior profitability, demonstrates that ACO can provide timely and efficient decisions in real-world deployments where both speed and scalability are critical.

Overall, these contributions establish ACO as a robust, profit-oriented, and scalable optimization framework for MEC–C-RAN systems, outperforming traditional GA-based, PSO-based, and heuristic approaches under realistic deployment conditions.

9. Conclusion

This paper analyzed computational offloading and resource optimization in MEC integrated with C-RAN, focusing on profit maximization for network operators. The performance of four algorithms—ACO, NGA, FGA, and Modified SJOORA—was evaluated under varying user densities. The results revealed that ACO consistently delivered the highest profitability, particularly in high-user scenarios, due to its superior optimization of power consumption, computational capacity, and fronthaul link usage. Power consumption increased nearly linearly with the number of users, reaching its peak at higher densities, highlighting the need for scalable energy management solutions. Computational capacity demands also rose significantly as user load increased, with notable jumps observed between 50 and 60 users. Similarly, fronthaul link capacity exhibited a steady upward trend, reflecting the growing data transmission requirements in high-density environments. Execution time analysis demonstrated that ACO achieved efficient performance with moderate scaling, making it suitable for real-time applications. In contrast, FGA showed the lowest execution time, averaging around 4 seconds, due to its efficient regression-based resource estimation. While NGA provided stable execution times (70 seconds), it lacked scalability in high-user scenarios. Modified SJOORA's execution time increased with user density, making it less effective for large-scale deployments. These findings underline the importance of balancing power consumption, computational capacity, and fronthaul link utilization with execution efficiency to optimize network performance. Future investigations will aim to enhance the adaptability of the proposed framework to real-time and dynamically changing user requirements. Moreover, hybrid optimization strategies will be explored to strengthen scalability and improve energy efficiency within MEC environments. Additional efforts will include the use of parallel computing to broaden the coverage area and support a larger number of users, alongside the integration of artificial intelligence techniques to further advance scalability and optimization efficiency.

10. Patents

Author Contributions: The concept for the experiment was devised collaboratively by all authors. S.M.A. and E.M.E. conducted the measurements and processed the data, while S.M.A. executed the numerical simulations. The manuscript was drafted by S.M.A. with input from all authors, who also participated in discussions and contributed to the interpretation of the findings.

Data Availability Statement: The datasets created and/or examined in this study can be obtained from the corresponding author upon a reasonable request.

Conflicts of Interest: The authors confirm that no conflicts of interest are associated with this work.

References

1. Chen, C.; Li, X.; Ji, H.; Zhang, H. Energy-efficient mobile edge computing system based on full-duplex energy harvesting relay network. *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)* 2020, **2020**, 1–6. https://doi.org/10.1109/GLOBECOM42002.2020.9348248
2. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.*, **2017**, *19*, 2322–2358. https://doi.org/10.1109/COMST.2017.2745201
3. Wang, Y.; Ge, H.; Feng, A.; Li, W.; Liu, L.; Jiang, H. Computation offloading strategy based on deep reinforcement learning in cloud-assisted mobile edge computing. *Proc. IEEE Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA) 2020,* **2020**, 108–113. https://doi.org/10.1109/ICCCBDA49378.2020.9095689
4. Liu, Y.; Li, Y.; Niu, Y.; Jin, D. Joint optimization of path planning and resource allocation in mobile edge computing. *IEEE Trans. Mob. Comput.* **2019**, *19*, 2129–2144. https://doi.org/10.1109/TMC.2019.2922316
5. Wang, K.; Yang, K.; Magurawalage, C.S. Joint energy minimization and resource allocation in C-RAN with mobile cloud. *IEEE Trans. Cloud Comput.* **2018**, *6*, 760–770. https://doi.org/10.1109/TCC.2016.2522439

6. Datta, J.; Das, A.; Khanra, S.; Chakraborty, S.; Sen, D. Compressive sensing based uplink C-RAN channel estimation with deep learning-aided optical fronthaul compensation. *Proc. Int. Conf. Converg. Technol. (I2CT) 2021,* **2021,** 1–4. https://doi.org/10.1109/I2CT51068.2021.9418196
7. Tout, H.; Mourad, A.; Kara, N.; Talhi, C. Multi-persona mobility: Joint cost-effective and resource-aware mobile-edge computation offloading. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1408–1421. https://doi.org/10.1109/TNET.2021.3066558
8. Zheng, R.; Xu, J.; Wang, X.; Liu, M.; Zhu, J. Service placement strategies in mobile edge computing based on an improved genetic algorithm. *Pervasive Mob. Comput.* **2024**, 101986. https://doi.org/10.1016/j.pmcj.2024.101986
9. Lin, H.; Zeadally, S.; Chen, Z.; Labiod, H.; Wang, L. A survey on computation offloading modeling for edge computing. *J. Netw. Comput. Appl.* **2020**, *169*, 102781. https://doi.org/10.1016/j.jnca.2020.102781
10. Tunga, H.; Kar, S.; Giri, D. Intrinsic profit maximization of the offloading tasks for mobile edge computing with fixed memory capacities and low latency constraints using ant colony optimization. *Math. Model. Eng. Probl*. **2022**, *9*, 668–674. https://doi.org/10.18280/mmep.090313
11. Spatharakis, D.; Dimolitsas, I.; Dechouniotis, D.; Papathanail, G.; Fotoglou, I.; Papadimitriou, P.; Papavassiliou, S. A scalable edge computing architecture enabling smart offloading for location based services. *Pervasive Mob. Comput.* **2020**, *67*, 101217. https://doi.org/10.1016/j.pmcj.2020.101217
12. Vambe, W.T.; Sibanda, K. A fog computing framework for quality of service optimisation in the Internet of Things (IoT) ecosystem. *Proc. Int. Multidiscip. Inf. Technol. Eng. Conf. (IMITEC) 2020*, **2020**, 1–8. https://doi.org/10.1109/IMITEC50163.2020.9334083
13. He, Z.; Xu, Y.; Liu, D.; Zhou, W.; Li, K. Energy-efficient computation offloading strategy with task priority in cloud-assisted multi-access edge computing. *Future Gener. Comput. Syst. 2023*, **2023**, *148*, 298-313. https://doi.org/10.1016/j.future.2023.06.014
14. Acheampong, A.; Zhang, Y.; Xu, X. A parallel computing based model for online binary computation offloading in mobile edge computing. *Comput. Commun.* **2023**, *203*, 248–261. https://doi.org/10.1016/j.comcom.2023.03.004
15. Pan, L.; Liu, X.; Jia, Z.; Xu, J.; Li, X. A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing. *IEEE Trans. Cloud Comput.* **2021**, *11*, 1334–1351. https://doi.org/10.1109/TCC.2021.3132175
16. Ge, H.; Geng, J.; An, Y.; Feng, H.; Zhou, T.; Huang, C. Research on collaborative computational offload strategy based on improved ant colony algorithm in edge computing. *Proc. Int. Conf. Nat. Lang. Process. (ICNLP) 2023,* **2023**, 486–490. https://doi.org/10.1109/ICNLP58431.2023.00093
17. Hussein, M.K.; Mousa, M.H. Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access* **2020**, *8*, 37191–37201. https://doi.org/10.1109/ACCESS.2020.2975741
18. An, X.; Li, Y.; Chen, Y.; Li, T. Joint task offloading and resource allocation for multi-user collaborative mobile edge computing. *Comput. Netw.* **2024**, *250*, 110604. https://doi.org/10.1016/j.comnet.2024.110604
19. Leguizamon, G.; Michalewicz, Z. A new version of ant system for subset problems. Proc. Congr. Evol. Comput. (CEC) **1999**, *2*, 1459–1464. https://doi.org/10.1109/CEC.1999.782655
20. Fidanova, S. Ant colony optimization for multiple knapsack problem and model bias. *Proc. Int. Conf. Numer. Anal. Appl.* 2004, **2004**, *3401*, 280–287. https://doi.org/10.1007/978-3-540-31852-1_33
21. Khan, P.W.; Abbas, K.; Shaiba, H.; Muthanna, A.; Abuarqoub, A.; Khayyat, M. Energy efficient computation offloading mechanism in multi-server mobile edge computing—An integer linear optimization approach. *Electronics* **2020**, *9*, 1010. https://doi.org/10.3390/electronics9061010
22. Guo, Y.; Zhao, Z.; Zhao, R.; Lai, S.; Dan, Z.; Xia, J.; Fan, L. Intelligent offloading strategy design for relaying mobile edge computing networks. *IEEE Access* **2020**, *8*, 35127–35135. https://doi.org/10.1109/ACCESS.2020.2972106

23. Bao, W.; Ji, H.; Zhu, X.; Wang, J.; Xiao, W.; Wu, J. ACO-based solution for computation offloading in mobile cloud computing. *Big Data & Information Analytics* **2015**, *1(1)*, 1–13. https://doi.org/10.3934/bdia.2016.1.1
24. Wang, Z.; Li, P.; Shen, S.; Yang, K. Task offloading scheduling in mobile edge computing networks. *Procedia Computer Science* **2021**, *184*, 322–329. https://doi.org/10.1016/j.procs.2021.03.041
25. Wang, K.; Yang, K. Power-minimization computing resource allocation in mobile cloud-radio access network. *In Proceedings of the 2016 IEEE International Conference on Computer and Information Technology (CIT), Nadi, Fiji,* **8–10 December 2016**, 667–672. https://doi.org/10.1109/CIT.2016.64
26. Sun, Y.; Wei, T.; Li, H.; Zhang, Y.; Wu, W. Energy-efficient multimedia task assignment and computing offloading for mobile edge computing networks. *IEEE Access* **2020**, *8*, 36702–36713. https://doi.org/10.1109/ACCESS.2020.2973359
27. Zhang, S.; Yi, N.; Ma, Y. Correlation-based device energy-efficient dynamic multi-task offloading for mobile edge computing. *In Proceedings of the 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), Helsinki, Finland,* **25–28 April 2021**, 1–5. https://doi.org/10.1109/VTC2021-Spring51267.2021.9448864
28. Guo, E.; Gao, Y.; Hu, C.; Zhang, J. A hybrid PSO-DE intelligent algorithm for solving constrained optimization problems based on feasibility rules. *Mathematic*s **2023**, *11(3)*, 522. https://doi.org/10.3390/math11030522
29. Chen, Z.; Hu, J.; Chen, X.; Hu, J.; Zheng, X.; Min, G. Computation offloading and task scheduling for DNN-based applications in cloud-edge computing. *IEEE Access* **2020**, *8*, 115537–115547. https://doi.org/10.1109/ACCESS.2020.3004509
30. Wang, Z.; Pei, Y.; Li, J. A survey on search strategy of evolutionary multi-objective optimization algorithms. *Applied Sciences* **2023**, *13*(7), 4643. https://doi.org/10.3390/app13074643
31. Kuri-Morales, A.F.; Gutiérrez-García, J. Penalty function methods for constrained optimization with genetic algorithms: A statistical analysis. *Mex. Int. Conf. Artif. Intell.* **2002**, *2313*, 108–117. https://doi.org/10.1007/3-540-46016-0_12
32. Singh, S.; Kim, D.H. Profit optimization for mobile edge computing using genetic algorithm. *IEEE Reg. 10 Symp. (TENSYMP)* **2021**, 1–6. https://doi.org/10.1109/TENSYMP52854.2021.9550947
33. Huang, X.; Leng, S.; Maharjan, S.; Zhang, Y. Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks. *IEEE Trans. Veh. Technol.* **2021**, *70(9)*, 9282–9293. https://doi.org/10.1109/TVT.2021.3096928
34. Gao, Z.; Yang, L.; Dai, Y. Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing. *IEEE Trans. Mob. Comput.* **2022**, *22(6)*, 3425–3443. https://doi.org/10.1109/TMC.2022.3141080
35. Zhang, J.; Wu, M.; Zhao, M. Joint computation offloading and resource allocation in C-RAN with MEC based on spectrum efficiency. *IEEE Access* **2019**, *7*, 79056–79068. https://doi.org/10.1109/ACCESS.2019.2922702
36. Fooladivanda, D.; Rosenberg, C. Joint resource allocation and user association for heterogeneous wireless cellular networks. *IEEE Trans. Wirel. Commun.* **2012**, *12(1)*, 248–257. https://doi.org/10.1109/TWC.2012.121112.120018
37. Wang, Y.; Han, Z. Ant colony optimization for traveling salesman problem based on parameters optimization. *Appl. Soft Comput.* **2021**, *107*, 107439. https://doi.org/10.1016/j.asoc.2021.107439
38. Wu, L.; Huang, X.; Cui, J.; Liu, C.; Xiao, W. Modified adaptive ant colony optimization algorithm and its application for solving path planning of mobile robot. *Expert Syst. Appl.* **2023**, *215*, 119410. https://doi.org/10.1016/j.eswa.2022.119410
39. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2015**, *24(5)*, 2795–2808. https://doi.org/10.1109/TNET.2015.2487344
40. Singh, S.; Kim, D.H. Joint optimization of computation offloading and resource allocation in C-RAN with mobile edge computing using evolutionary algorithms. *IEEE Access* **2023**, *11*, 112693–112705. https://doi.org/10.1109/ACCESS.2023. 3322650