Received 30 August 2023, accepted 26 September 2023, date of publication 6 October 2023, date of current version 18 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3322650

THEORY

Joint Optimization of Computation Offloading and Resource Allocation in C-RAN With Mobile Edge Computing Using Evolutionary Algorithms

SUMIT SINGH[1], **AND DONG HO KIM**[2], **(Senior Member, IEEE)**
[1]Department of Integrated IT Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea
[2]Department of IT Media Engineering, Seoul National University of Science and Technology, Seoul 01811, Republic of Korea

Corresponding author: Dong Ho Kim (dongho.kim@seoultech.ac.kr)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) funded by the Korean Government [Ministry of Science and ICT (MSIT)] [Development of the 6G Service Targeted AI/ML-Based Autonomous-Regulating Medium Access Control (6G STAR-MAC)] under Grant 2021-0-00368.

ABSTRACT Mobile Edge Computing has been widely recognized as a key enabler for new latency-sensitive applications on resource constrained mobile devices. The objective to offload a computationally intensive task to a cloud server is in general intended to reduce the system's energy consumption and/or latency. In this paper, we attempt to examine how profitable computation offloading is from a service provider's perspective. The joint optimization of radio and computing resources along with offloading decisions results in a mixed integer nonlinear optimization problem which belongs to the class of NP-hard problems. To counter this challenge, we decouple the offloading decision from the resource allocation problem. Initially, approximately optimal offloading decisions are determined using evolutionary algorithms such as genetic algorithms and binary particle swarm optimization algorithms. After several iterations of the evolutionary process to make offloading decisions, the optimal solution is ultimately obtained that performs resource allocation based on exact calculation of the profit value. For faster execution of the evolutionary algorithm, instead of using an optimization solver to find the exact solution, we use a novel approach to seeding the initial population and a regression-based machine learning method to predict the optimal resource allocation values to minimize the objective function evaluation time. According to the simulations performed as part of this study, the proposed evolutionary algorithms outperform existing spectral efficiency-based offloading algorithm in terms of profitability, with shorter execution times as well. The effects of resource availability and the parameters of the algorithm on the profitability of offloading are also examined.

INDEX TERMS Computation offloading, genetic algorithm, binary PSO, profit in MEC.

I. INTRODUCTION

As latency-sensitive and computationally intensive services such as AR-VR and autonomous vehicles are emerging, the demand for computation capabilities in mobile devices has increased tremendously. Also, because the battery capacity of these devices grows only at a rate of nearly 5-6% every year [1], supporting these services becomes even more difficult. The main constraints of mobile computing are limited energy, low computation capabilities and limited wireless bandwidth, all of which require mobile devices to offload computationally intensive tasks to nearby servers. Researchers have examined whether or not such offloading can save energy [2]. Broadly, offloading mechanisms can be divided into two types. In the first type, mobile terminals determine whether to offload or not by comparing the energy consumption of executing a task locally with the energy consumption of sending data to an edge server for processing [3]. In the other type of offloading, the edge server has a central control unit that contains all task information

The associate editor coordinating the review of this manuscript and approving it for publication was Hassan Omar.

and the channel state information (CSI) of all connected user equipment (UEs). It then decides which UEs will offload tasks based on a task partitioning framework [4] and on the joint optimization of energy consumption under latency and other resource constraints [5], [6].

The cloud radio access network (C-RAN) [7] and mobile edge computing (MEC) [8], [9] are two exciting cloud technologies which can play a pivotal role in supporting upcoming services on mobile terminals. C-RAN is a form of distributed base station deployment in which there is a central base band unit (BBU) pool and many remotely connected remote radio heads (RRHs). In cloud RAN, the BBU pool is responsible for PHY baseband processing and MAC layer processing and requires strong computing resources. The RRHs transmit and receive the signals from BBU pool via a fronthaul link. It also communicates these signals to mobile users on the RF frequency band. On the other hand, MEC brings the concept of cloud computing to the cellular network in order to handle computationally intensive and latency-sensitive applications. So, it is necessary to decide whether to offload the tasks generated by the application to a nearby edge server to save the terminal's battery instead of processing them locally. In the efforts to achieve an efficient offloading strategy, joint optimization of radio and computing resources remains a challenge because such resource allocation problems often tend to be of the NP-hard type. In [10], the authors presented a survey of the MEC research literature related to joint optimization of resource allocation. As the system becomes more complex, solving an optimization problem for energy efficiency or latency of the overall system becomes computationally demanding. Therefore, a distributed optimization algorithm for multi-cell MEC was developed [11]. In this paper, however, we confine our discussion to the formulation of a strategy for computation offloading so as to maximize the network operator's profit. The reader interested in computation offloading modeling and approaches to solve such resource allocation problems is referred to an earlier study [12] for a comprehensive survey of the topic.

A. RELATED WORKS AND OUR CONTRIBUTIONS

Researchers have formulated a multi-objective optimization problem for MEC, taking into account user satisfaction, profit of the network operator and resource utilization of cloudlets [13]. The authors in [14] proposed a multi-objective evolutionary algorithm based on a genetic algorithm, and suggested applying the statistics-based Taguchi technique to determine the parameter values of the genetic algorithm. In other work [15], the authors used migrating bird [16] and simulated an annealing based algorithm for profit optimization. The modeled optimization problem was nonlinear, which was then converted to an unconstrained optimization problem using penalty terms. In [17], the authors formulated a non-convex joint optimization problem of service caching and computation offloading to maximize profits in MEC. The optimization problem was difficult to solve, so the authors designed an algorithm which used Lyapunov optimization theory to transform the original long-term problem into a slot-by-slot optimization problem that can be solved by knowing only the current slot information. Other authors formulated a profit maximization problem in MEC with the C-RAN scenario and proposed the spectrum-efficiency-based joint optimization of offloading and resource allocation processes [18].

MEC will become a key pillar of 5G and 6G networks to support ultra-reliable and low-latency communications (URLLC). Further, computation offloading is one of the important use cases of MEC. Based on a literature survey focusing on MEC, we define the problem of the joint optimization of resource allocation and offloading decisions where the objective is to maximize the profit of the network service provider in a C-RAN scenario.

- Using a system model and problem formulation proposed earlier [18], we adopt a different solution strategy. Instead of relying solely on the spectrum efficiency of the RRH-UE links for offload decisions, we add task-related information to the spectrum efficiency to create a ranking. UEs with higher ranking are likely to generate more revenue for the service provider.
- We propose solutions based on a genetic algorithm and particle swarm optimization for the joint optimization problem of finding computation offloading strategy and resource allocation method. For the rapid execution of the evolutionary algorithms, we use a machine learning based regression model to predict the resource allocation values during the evolutionary cycle. Also, we compare the proposed algorithms with other existing algorithms to demonstrate their superior performance. The proposed algorithms have nearly a constant execution time which does not increase with an increase in the number of UEs, making them ideal scalable solutions.
- Lastly, the impact of resource availability on profitability of computation offloading is examined.

II. SYSTEM MODEL

We consider a scenario in which a MEC server is collocated at the BBU pool of a C-RAN. The RRHs are connected to the BBU pool via fronthaul links. The mobile devices are connected to RRH via wireless links. We assume that the computation capacity of each mobile device is limited. Thus, the MEC server allocates computational and radio resources for the mobile device's computational processing. The offloading scenario assumes a task-aware approach, which means that the offloading strategy is determined by considering the types and characteristics of the task. For the offloading service, the UEs must send a prior indication signal to the MEC server, which then charges the UEs for caching, processing and sending the results. If the MEC server does not accept a UE's request, the UE then executes the task using its local CPU. Accordingly, no revenue is earned by the service provider in this case. Hence, the MEC

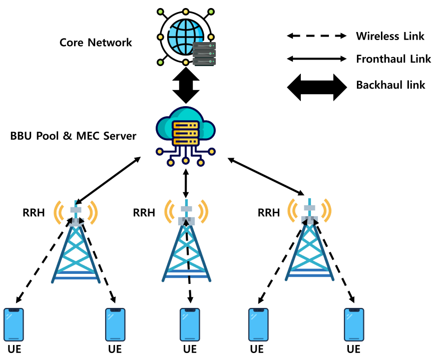


FIGURE 1. MEC in C-RAN system.

service provider must choose a combination of UE tasks to maximize its profit.

In the C-RAN structure, let $\mathcal{N} = \{1, 2, \ldots, N\}$ be the set of RRHs connected to the BBU pool through fronthaul links. Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be the set of UEs accessing the network through these RRHs. For simplicity, we assume that each UE and RRH is equipped with only a single antenna. The offloading matrix is $\mathbf{A} = \{a_{m,n}\}_{M\times N}, \forall m \in \mathcal{M}, n \in \mathcal{N}$, where a binary variable $a_{m,n}$=1 implies that UE m is permitted to access the MEC server through RRH n, and 0 if not so. Let $\mathcal{M}^p$ be the set of UEs that are permitted to offload such that $\mathcal{M}^p \subset \mathcal{M}$. Considering the task awareness case, we assume that the task data of the UEs has already been cached into the MEC server. We denote a task W_m of UE m by

$$\mathbf{W_m} = (F_m, D_m, T_{m,max}), \quad m = 1, 2 \ldots, M \tag{1}$$

where F_m denotes the number of CPU cycles required to accomplish the task, D_m represents the output data size to be transmitted to UE m after the execution of the task on the MEC server via RRH, and $T_{m,max}$ is the time limit within which the task execution result should finally reach UE m.

A. COMMUNICATION MODEL

It is assumed that the MEC server has access to full downlink channel state information (CSI) from all RRHs to all UEs. We consider a frequency reuse in multi-cell environment where the spectrum used by RRH is overlaid, thus causing inter-RRH interference. Intra-RRH interference is ignored here because the spectrum is assigned orthogonally at the RRH. The signal to interference plus noise ratio (SINR) for RRH n transmitting to UE m is given as:

$$\text{SINR}_{m,n} = \frac{p_n g_{m,n}}{\sigma^2 + \sum_{j=1,j\neq n}^{N} p_j g_{m,j}} \tag{2}$$

where p_n is the transmit power of RRH n; $g_{m,n}$ represents the channel gain from RRH n to UE m; σ^2 is defined as the variance of Additive White Gaussian Noise distributed normally as $\mathcal{N}(0, \sigma^2)$. According to Shannon's channel capacity formula, the spectrum efficiency of UE m and RRH n link is given by:

$$e_{m,n} = \log_2 \left(1 + \frac{p_n g_{m,n}}{\sigma^2 + \sum_{j=1,j\neq n}^{N} p_j g_{m,j}}\right). \tag{3}$$

The total spectrum bandwidth that can be allocated at each RRH is B Hz. The fraction of bandwidth that the RRH allocates to an UE is represented by an element of matrix $\mathbf{b}$=$\{b_{m,n}\}_{M\times N}, \forall m \in \mathcal{M}, n \in \mathcal{N}$ and further the sum of total bandwidth fractions allocated at any RRH should be less than one, i.e., $\sum_{m\in\mathcal{M}^p} b_{m,n} \leq 1, \forall n \in \mathcal{N}$. The achievable instantaneous rate of UE m accessing RRH n can then be calculated as:

$$R_{m,n} = a_{m,n} b_{m,n} B e_{m,n}. \tag{4}$$

Summing up the instantaneous rates offered by each RRH to its client UEs gives the fronthaul capacity constraint,

$$\sum_{m\in\mathcal{M}^p} R_{m,n} \leq L_n, \forall n \in \mathcal{N} \tag{5}$$

where L_n is the fronthaul capacity for RRH n. The time taken by the MEC to send the output data back to UE m after execution is given by

$$T_m^{\text{Tr}} = \frac{D_m}{\sum_{n\in\mathcal{N}} R_{m,n}} \tag{6}$$

where $\sum_{n\in\mathcal{N}} R_{m,n}$ denotes the combined transmission rate obtained from summation for all RRHs supporting offloading to the UE m. We ignore the transmission time from MEC to RRH as it is negligible compared transmission time from RRH to the UE. The task uploading transmission time is also ignored, as we assume that MEC has task awareness and that the task data are already cached in the MEC server.

B. COMPUTATION MODEL

Once it is determined which UEs are permitted to execute the tasks in the edge cloud, the MEC server then allocates the computing resources to the permitted UEs (in terms of CPU cycles/s). Let F denote the total computing resources available to the MEC server. Then, we define $c_m \in [0, 1], \forall m \in \mathcal{M}^p$ as a fraction of the computing resources (F) allocated by the MEC server to UE m, which have a constraint of $\sum_{m\in\mathcal{M}^p} c_m \leq 1$. This results in computing resource allocation set $\mathbf{c}$=$\{c_m\}, \forall m \in \mathcal{M}^p$. At this point, we can calculate the execution time for task $\mathbf{W_m}$ on the MEC server with

$$T_m^{\text{Exe}} = \frac{F_m}{c_m F} \tag{7}$$

where F_m is the required number of CPU cycles for the execution of task $\mathbf{W_m}$ and $c_m F$ is the computing resource in cycles/s allocated to UE m by the MEC server. Hence, the total time for task offloading process for UE m is determined by the sum of the execution time in the cloud and the transmission time back to the UE, as

$$T_m = T_m^{\text{Exe}} + T_m^{\text{Tr}}. \tag{8}$$

Also, for offloading to be useful to UEs, the amount of computing resources allocated to a UE in the cloud should be greater than its local computing resource (f_m^{Local}), resulting in the constraint

$$c_m F \geq a_{m,n} f_m^{\text{Local}}. \tag{9}$$

C. PROFIT FUNCTION

The goal of this study is to maximize the profits of network operators. Therefore, we need to model the revenue and cost functions to arrive at the profit objective function. We define the unit price charged per CPU cycle in the cloud as p_f, the fee for caching the original task data related to the size of task data as S_m, and the unit price per bit for data transmission to the UE as p_t, respectively. The revenue that can be obtained from offloading the task $\mathbf{W_m}$ for a permitted UE m is then given as

$$\Omega_m = p_f F_m + p_t D_m + S_m. \tag{10}$$

Taking cost into account, the unit price of spectrum assigned to a UE is q_b per Hz and the unit price of the computing resource is q_c per CPU cycle/second. Finally, considering the revenue and cost functions, the profit from an offloading-permitted UE m is

$$U_m = \Omega_m - \kappa c_m F q_c - \omega \sum_{n \in \mathcal{N}} b_{m,n} B q_b \tag{11}$$

where ω and κ denote the impact coefficients which represent the tradeoff between scarcity and price fluctuation of the spectrum resources and computing resources, respectively.

For simplicity, we define $\Gamma_m^C = \kappa F q_c$ and $\Gamma_m^T = \omega B q_b$, with the the overall profit function from all offloading-permitted UEs then defined as

$$U = \sum_{m \in \mathcal{M}^p} \left(\Omega_m - c_m \Gamma_m^C - \sum_{n \in \mathcal{N}} b_{m,n} \Gamma_m^T \right). \tag{12}$$

Therefore, our profit optimization problem can be written as

$$\begin{aligned}
\max_{a_{m,n}, b_{m,n}, c_m} \quad & \sum_{m \in \mathcal{M}^p} \left(\Omega_m - c_m \Gamma_m^C - \sum_{n \in \mathcal{N}} b_{m,n} \Gamma_m^T \right) \\
\text{s.t. } C1: & \sum_{m \in \mathcal{M}^p} a_{m,n} b_{m,n} - 1 \leq 0, \forall\, n \in \mathcal{N}, \\
C2: & \sum_{m \in \mathcal{M}^p} \sum_{n \in \mathcal{N}} a_{m,n} c_m - 1 \leq 0, \\
C3: & T_m^{\text{Tr}} + T_m^{\text{Exe}} - T_{m,max} \leq 0, \forall m \in \mathcal{M}^p, \\
C4: & \sum_{\forall n \in \mathcal{N}} a_{m,n} f_m^{\text{Local}} - c_m F \leq 0, \forall m \in \mathcal{M}^p, \\
C5: & \sum_{m \in \mathcal{M}^p} R_{m,n} - L_n \leq 0, \forall n \in \mathcal{N}, \\
C6: & a_{m,n} \in \{0, 1\}, \forall n \in \mathcal{N}, \forall m \in \mathcal{M}^p.
\end{aligned} \tag{13}$$

The optimization problem above is a mixed integer non-linear program which belongs to the class of NP-hard problems in general. However, if we separate the offloading strategy from the resource allocation problem, (13) is reduced to a simple linear program problem. Therefore, first we attempt to find an approximately optimal offloading matrix $\mathbf{A}$ using evolution based methods and then use it as the input to the aforementioned optimization problem to find the optimal $\mathbf{b}*$ and $\mathbf{c}*$. This is illustrated in Figure 2.

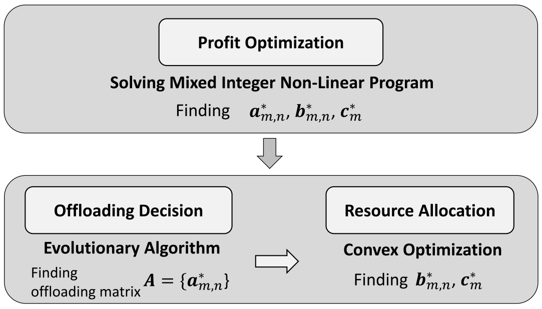

Profit Optimization
Solving Mixed Integer Non-Linear Program
Finding $a^*_{m,n}, b^*_{m,n}, c^*_m$
Offloading Decision
Evolutionary Algorithm
Finding offloading matrix $A = \{a^*_{m,n}\}$
Resource Allocation
Convex Optimization
Finding $b^*_{m,n}, c^*_m$

FIGURE 2. Overview of the solution strategy.

III. GENETIC ALGORITHM-BASED OFFLOADING STRATEGY

A Genetic Algorithms (GA) is a type of Evolutionary Algorithm [19] based on the concept of evolution through natural selection, which is suitable for many types of problems and can provide very good (not necessarily optimal) solutions to difficult problems in a reasonable amount of time. A GA is characterized by binary representation of individuals, fitness proportionate selection, a low mutation probability, and genetically inspired recombinations.

In [18], the authors outlined a spectral efficiency-based joint optimization of radio and computing resources allocation(SJOORA) for determining the offloading strategy of (13). Here, we propose a genetic algorithm customized to the problem scenario under consideration to find an approximately optimal offloading strategy and compare its performance with the SJOORA algorithm. When applying a genetic algorithm that requires testing random populations for candidate solutions, it is desirable to be able to easily compute the fitness/objective functions for the solution. However, the objective function in (13) is computationally expensive to calculate. Using a typical genetic algorithm to find the best possible offloading strategy, it can take a long time to compute the optimal value of (13) multiple times. This can severely constrain the MEC server, as computing resources must be allocated for finding the optimal offloading strategy instead of actually computing the offloaded tasks. As a tradeoff between fast computation and finding an optimum solution, we propose a fast genetic algorithm that is suitable for this problem (13).

A. SIMPLE GENETIC ALGORITHM

For comparison with the proposed fast genetic algorithm, we first describe a general method for applying a genetic algorithm to (13), which we call the Simple Genetic Algorithm (SGA) (See Fig. 3)

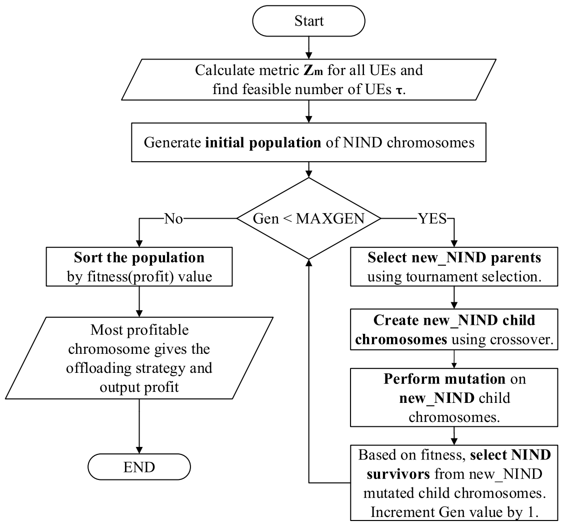

Start
Calculate metric Zm for all UEs and find feasible number of UEs τ.
Generate initial population of NIND chromosomes
Gen < MAXGEN
No
YES
Sort the population by fitness(profit) value
Most profitable chromosome gives the offloading strategy and output profit
END
Select new_NIND parents using tournament selection.
Create new_NIND child chromosomes using crossover.
Perform mutation on new_NIND child chromosomes.
Based on fitness, select NIND survivors from new_NIND mutated child chromosomes. Increment Gen value by 1.

FIGURE 3. A flowchart illustrating genetic algorithm.

First, the MEC server ranks each UE that requests an offload service. This ranking is based on the computational and communication requirements of the UE tasks and the best achievable link spectral efficiency between the UE and all RRHs. The process of ranking is as follows: First, we create a task metric of UE m, $\lambda_\mathbf{m} = (\lambda_{m_1}, \lambda_{m_2}, \lambda_{m_3}) = (\alpha_1 F_m, \alpha_2 D_m, \alpha_3 T_{m,max})$ and get a metric value $\Lambda_m = \sum_{i=1}^{3} \lambda_{m_i}$ of the task $\mathbf{W_m}$, where α_i's are the weighting coefficients of the λ_{m_i}. We combine the elements of $\lambda_\mathbf{m}$ linearly because the higher the computational requirement, output data size, and latency constraints, the more revenue is generated for the network operator. Another factor to consider when ranking is the spectral efficiency of the links available to the UE. Let $e_m = \max(e_{m,n}), \forall n \in \mathcal{N}$ be best link efficiency available to UE m among all the RRHs. Let r_1 be the range of task metric Λ, i.e., $r_1 = \max(\Lambda) - \min(\Lambda)$ and r_2 be the range of max spectrum efficiencies e_m available to the UEs, i.e., $r_2 = \max(e_m) - \min(e_m), \ \forall m \in \mathcal{M}$. The metric Z_m for offload profitability of UE m is then given by

$$Z_m = \frac{r_1 \times e_m + r_2 \times \Lambda_m}{r_1 + r_2}. \tag{14}$$

UE tasks with more radio and computing resource requirements for execution, less strict latency demands and better spectral efficiency are ranked higher. For the SJOORA algorithm considered for performance comparison, each UE only offloads to the RRH that provides the best spectral efficiency.

Next, we find an approximate number of UEs whose tasks can be offloaded under the resource constraints of the MEC. Let this number be represented by τ. In other words, $\tau \approxeq \text{card}(\mathcal{M}^p)$. This can be obtained by repeatedly solving for the constraints in (13) using a cvx (SDPT3) solver [20], where during every iteration of the loop, the optimization constraints are solved for an increased number of UEs. The loop stops when the problem becomes infeasible for the current UE count. This number is used as the starting value for the total number of UEs permitted to offload in the initial population of the genetic algorithm (i.e., the number of ones in a binary string).

Using τ, we create an initial population of NIND chromosomes such that each chromosome has only τ number of ones, with the remaining $M - \tau$ entries being zeros, where NIND denotes the number of individuals in the population. Note that the optimal solutions are expected to contain more high-ranked UEs. Hence, we seed some individuals in the initial population so that the number of highest-ranked UEs in the chromosome has some fraction of τ. The remaining individuals are generated by random sampling so that each individual has τ number of permitted UEs, where the probability of the selection of a UE is directly proportional to its rating.

Following the steps above, we now have the total number of NIND individuals to start the evolutionary process of the genetic algorithm. The following steps are repeated for each generation until we reach the MAXGEN generations:

1) First, we check the conditions for changing the mutation rate and whether to terminate the evolutionary cycle or not. Refer to the simulation code in earlier work [21].
2) Next, for each chromosome given as input to the offloading decision, we find the objective (profit) value by solving (13). We normalize the objective values of all chromosomes to generate a fitness vector.
3) We find the number of new individuals (offspring) to be created from the existing population, i.e., new_NIND = GGAP×NIND, where GGAP is the generation gap having a value greater than 1. We create more individuals than the existing population.
4) Next, we perform parent selection using a tournament selection method [19].
5) We create new_NIND child chromosomes from a pool of new_NIND parents using a recombination process (Fig. 4).
6) For these new_NIND child chromosomes, we perform a mutation operation, i.e., the flipping of bits at random locations. The fraction of bits flipped is controlled by a parameter called the mutation ratio (MUTR).
7) The objective value (profit) of each mutated new_NIND chromosomes is evaluated and the best NIND individuals among them are chosen as the resulting population for that generation.

In the final generation, the fittest individual, i.e., the chromosome giving the maximum profit value is taken as the output of the Simple Genetic Algorithm. Note that to calculate the profit for a chromosome, the offload decision matrix is derived by element-wise multiplying the transpose of the chromosome vector by the initial offload matrix. This initial offload matrix is identical to that in step 2 of Algorithm 1. The resulting offload decision matrix is fed as the input to the optimization problem (13), which can then be solved using an optimization solver.

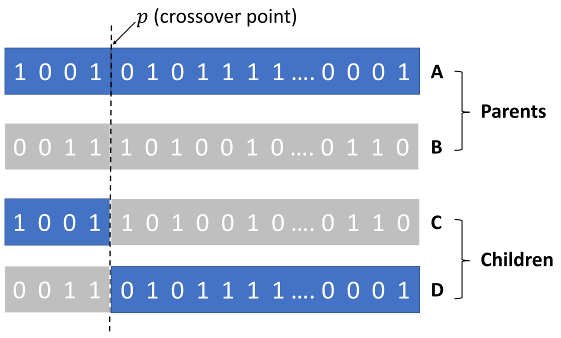

p (crossover point)
1 0 0 1 0 1 0 1 1 1 1 .... 0 0 0 1 A
0 0 1 1 1 0 1 0 0 1 0 .... 0 1 1 0 B
Parents
1 0 0 1 1 0 1 0 0 1 0 .... 0 1 1 0 C
0 0 1 1 0 1 0 1 1 1 1 .... 0 0 0 1 D
Children

FIGURE 4. **Recombination process.**

B. FAST GENETIC ALGORITHM

The Simple Genetic Algorithm described in subsection III-A requires much computation and takes longer time to execute because the objective function is a constrained optimization problem. Since computational offloading in MEC is usually for delay-sensitive tasks, this is not a favorable scenario towards a better solution. Therefore, we propose a statistics-based fast genetic algorithm to take full advantage of the genetic algorithm.

The Fast Genetic algorithm (FGA) follows the same evolutionary steps as the SGA, except that the objective value (profit) calculation is different. In the FGA, we define a new objective function by adding penalties for constraint violations to the objective function of the optimization problem (13). Constraints in the optimization problem can also be managed with other advanced methods [22], but we prefer to use the penalty function approach for its simplicity and faster execution. The penalty coefficients for constraint violations in penalty functions were chosen empirically such that the overall penalty for constraint violations can be of the order of the objective function and to discourage any infeasible solutions. By using this technique, we can afford to start with a larger initial population and repeat the evolutionary process for many generations to arrive at a better final population in much less time as compared to the simple genetic algorithm with the same parameter values. At the end of the evolutionary process, we solve the optimization problem (13) with the best of l individuals with highest fitness in the final population, and select the most profitable one among them as the final offloading strategy. l can be set to a small value such as 3 or 5. Discussions of FGA can be summarized in the form of Algorithm 1.

In section III-A, we used a convex optimization solver to calculate the profit for a chromosome. However, the "CalculateProfit" function in the FGA Algorithm 1 does not use the solver. Instead, it calculates the sum of the profit and penalty for a given chromosome. The amount of penalty is proportional to the number of constraint violations.

To calculate the profit with the "CalculateProfit" function in Algorithm 1, we need some approximate values of radio resource allocation $\mathbf{b}^*$ and computing resource allocation $\mathbf{c}^*$ for the given offloading strategy $\mathbf{A}$. We use a decision tree regression-based Machine Learning model in MATLAB to predict the values of $\mathbf{b}^*$ and $\mathbf{c}^*$. To build the regression model, we need a set of optimization-related dataset. To obtain that data, we solved the optimization problem (13) multiple times with random offloading strategies and recorded the following information for each strategy:

- Total number of UEs which were permitted to offload
- Total number of RRHs which support at least one UE
- Total fraction of the bandwidth allocated to UEs by each RRH
- Total number of UEs supported at each RRH
- Total fraction of computational resources allocated by the MEC server

The generated dataset is filtered out to remove results that are infeasible. Next, we use the following approximations:

Algorithm 1 Fast Genetic Algorithm

```
Input: $e\ M\ N\ W\ B\ F$ NIND GGAP MUTR MAXGEN and
    other simulation parameters
Output: Profit, Offloading strategy $\mathcal{A}$
1:  Create $\mathbf{e_{max}}$ of length $M$ where $\mathbf{e_{max}} = \max_n(\mathbf{e})$, $\forall m \in M$.
2:  Create $\mathbf{A}_{in}$ where $a_{m,n_{in}} = 1$ if $n = \arg\max_n \mathbf{e} \forall m \in M$; 0 otherwise .
3:  Calculate $Z_m$ for ranking all UEs.
4:  Chrom ← Create an initial Population of NIND individuals.
5:  Gen ← 0
6:  while Gen < MAXGEN do
7:      Check the conditions for mutation ratio change and
        exiting out of evolution and update accordingly
8:      ObjV ← CalculateProfit(Chrom)
9:      FitnV ← Normalize(ObjV)
10:     new_NIND ← floor(GGAP × NIND)
11:     ParCh ← TournamentSelect(Chrom, FitnV, new_NIND)
12:     $i \leftarrow 1$
13:     while $i <$ new_NIND do
14:         [ChldCh[i], ChldCh[i+1]] ← CrossingOver(ParCh[i], ParCh[i+1])
15:         $i \leftarrow i + 2$
16:     end while
17:     for $i \leftarrow 1$ to GGAP×NIND do
18:         MutCh[i] ← Mutation(ChldCh[i], MUTR)
19:     end for
20:     ObjVMut ← CalculateProfit(MutCh)
21:     Chrom ← Selection(MutCh, ObjVMut, NIND)
22:     Gen ← Gen + 1
23: end while
24: ObjV ← CalculateProfit(Chrom)
25: Sort ObjV in descending order.
26: Calculate the profit for the top $l$ chromosomes with the
    highest profit using the optimization solver. Let a_best
    be the fittest chromosome out of these $l$ chromosomes.
27: Final offloading strategy $\mathbf{A} = a\_best^T \odot \mathbf{A}_{in}$.
```

1) We train a regression model on the generated dataset to predict the value of the total fraction of the bandwidth allocated at RRH n, $\sum_{m\in\mathcal{M}} b_{m,n}$, where the predictor variables are the total number of UEs permitted to offload, the number of RRHs supporting any of the UEs and the number of UEs supported by each RRH. The total fraction of bandwidth allocated at RRH n is then equally distributed among the number of UEs supported by it to find $b_{m,n}$.
2) We calculate the mean of computing resource allocation fractions for the entire dataset, mean_frac, and further consider the ratio of each UE's computation requirement F_m among all UEs as c_m = mean_frac $\times \frac{F_m}{\sum_{m\in\mathcal{M}^p} F_m}$. The allocation of the MEC's computing resources in proportion to the task requirements of UEs was done in earlier work [23].

So far, our "CalculateProfit" function has not considered constraint violations in its profit calculation. In order to compensate for violations, we include a penalty term in this calculation. The requirements which are set by the constraints (*C1, C2, C6*) are automatically met during the implementation of the"CalculateProfit" function. To decide a suitable penalty function for violation of constraints *C3, C4* and *C5* of (13), we compared our penalty function in earlier work [24], expressed as

$$\begin{aligned}\text{Penalty}_1 = -(\theta_1 \times (\max(\sum_{m\in\mathcal{M}} C3, 0))^2 \\ + \theta_2 \times \max(\sum_{m\in\mathcal{M}} C4 \times 10^{-10}, 0)^2 \\ + \theta_3 \times \max(\sum_{n\in\mathcal{N}} C5 \times 10^{-8}, 0)^2)\end{aligned} \tag{15}$$

with the penalty functions given by (3)-(6) in another study [25]. The θ_is are constants and the multiplier values, 10^{-10} and 10^{-8} in the penalty function, are selected such that they bring the constraint violations to the same numerical order. Fig. 5 shows a comparison of the different penalty functions (NIND = 30 and MAXGEN = 7, results averaged over ten iterations). The best penalty function was found to be

$$\begin{aligned}\text{Penalty}_3 = -(\phi_1 \times (\sum_{m\in\mathcal{M}} \max(C3, 0)) \\ + \phi_2 \times \sum_{m\in\mathcal{M}} \max(C4 \times 10^{-10}, 0) \\ + \phi_3 \times \sum_{n\in\mathcal{N}} \max(C5 \times 10^{-8}, 0))\end{aligned} \tag{16}$$

where the ϕ_is are constants.

The Penalty$_3$ was chosen as the penalty function for Algorithm 1 and was added to the profit calculated by the "CalculateProfit" function to determine the actual (penalized) profit value for a chromosome.

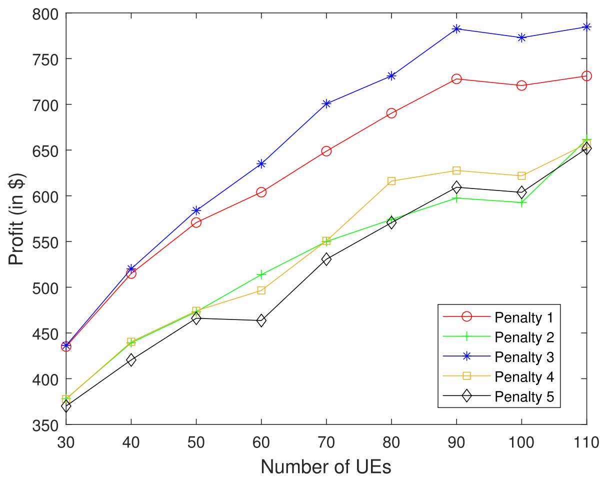


FIGURE 5. **Penalty functions comparison.**

Our previous work [21] explains the implementation of the Algorithm 1 using the MATLAB code.

C. MODIFIED SJOORA

When implementing the SJOORA algorithm as proposed in the aforementioned study [18], we found that the combination of UEs finally selected for offloading can occasionally be infeasible. Therefore, we considered a modified SJOORA algorithm in which we keep reducing the number of UEs until (13) becomes feasible for the selected combination of UEs.

D. GREEDY ALGORITHM

As another algorithm for performance comparison, we considered a greedy algorithm, which functions quite similarly to the initial population creation process in the SGA. We first find an approximate number of UEs, τ, which can be feasibly offloaded under the resource constraints, then sort the UEs according to five different offloading strategies based on rankings and select some number of UEs starting from the top. For example, for the first strategy, we select 95% of the feasible number of UEs starting from the highest ranked UE. Likewise, we do this for the remaining four offloading strategies, each having a different percentage value. We then calculate the profit of each strategy and select the most profitable one as the final offloading strategy for the greedy algorithm. Note that the greedy algorithm does not search for the optimal solution in any way. It merely performs an initial selection, assuming that higher-ranked chromosomes will yield a higher profit value.

We choose five different τ because choosing only the highest ranked τ UEs for offloading may result in an infeasible solution. On the other hand, a lower τ may be less profitable, but more feasible. Therefore the algorithm tries to find τ that results in the most profitable and feasible solution.

IV. PSO-BASED OFFLOADING STRATEGY

Particle Swarm Optimization (PSO) [26] is another class of evolutionary algorithm widely used for non-linear optimization problems. PSO is inspired by the social behavior, such as birds flocking or fish schooling, and differs from other evolutionary algorithms in that it does not use crossover with its mutation. Each particle is an object variable with various attributes, such as position, value, last position, probability vector, personal and global best positions and their corresponding values. PSO was originally introduced to continuous domain problems where a particle's trajectory is defined by a change in position, i.e., velocity, which depends on the difference between the personal best position, global best position and the particle's current position. Binary PSO (BPSO) is a modified version of PSO for the binary domain, and the position value is determined as 1 or 0. Trajectory (or velocity) that depends on the difference between the best position and the current position generate probability using a sigmoid function and determine whether the next position will be 1 or 0. This means that if the probability value is greater than a random number between [0,1] extracted from a uniform distribution, the next position is determined as 1 and, otherwise as 0. So, the greater the difference between the current position and the best position, the greater the probability that the position value will flip to 1 [27].

A. APPLYING SBPSO FOR PROFIT OPTIMIZATION

We apply the Sticky Binary Particle Swarm Optimization (SBPSO) algorithm introduced in [28] to solve (13), where stickiness is a value between 0 and 1 that represents the property of maintaining the current bit value. A stickiness is set to 1 as soon as the bit is flipped and decays linearly in the next iterations. In SBPSO, the greater the difference between the particle's current position and the best position and the smaller the stickiness, the more likely it is to flip. In one study [29], the authors proposed a dynamic version of SBPSO, namely DSBPSO, to control the algorithm parameters during the evolutionary process. They also compared the performance of these two algorithms with other state-of-the art algorithms, specifically the novel modification binary differential evolution (NMBDE) [30], Time-Varying BPSO [31], Up BPSO [32], and Quantum BPSO [33] algorithms for knapsack and feature selection problems, demonstrating the superior performance of these two algorithms (SBPSO and DSBPSO) in most cases. Another advantage of SBPSO and DSBPSO is their simple computation, which uses only basic mathematical operators. Therefore, here we use these algorithms to solve the profit optimization problem (13).

B. INITIALIZATION

We initialize with a fixed number of particles, P. Each particle's position is a $(1, M)$ binary string, where the m^{th} index is 1 if UE m is allowed to offload, and 0 otherwise. Each particle's position is equivalent to a chromosome in the genetic algorithm. The value (profit) of a position is calculated using the "CalculateProfit" function in Algorithm 1.

As was done in section III-A, we create rankings for UEs and find the initial number of feasible UEs τ. The positions of particles are initialized in the same way to generate the initial population. At the beginning of the first iteration, each particle's last position, personal best and global best positions are set equal to its current position. The probability vector of flipping a bit is a $(1, M)$ vector, which depends on three factors: (a) how recently the bit has been flipped, (b) its distance from the personal best and (c) its distance from the global best. In our scheme, flipping a bit corresponds to changing whether the m^{th} UE's task is offloaded or not. Each element of a probability vector is initialized randomly between 0 to 1. Parameter settings for the algorithms are available in the literature [29].

C. ITERATION PROCESS

For every particle, we evaluate the profit value of the particle's position based on whether the UEs' tasks are offloaded or not, at each iteration. If the value of the particle is greater than its personal best value, we then update the personal best value and the corresponding position. Also, if the value of the particle is greater than the global best value, we then update the global best value and position of all particles to the value and position of the current particle.

The elements of each particle's position vector and probability vector are also updated. For the position vector of particle i, in the $(t+1)^{\text{th}}$ iteration, if the bit value of UE m in the last iteration is flipped, we set the stickiness to 1, otherwise, we update it according to

$$\text{stkns}_{i,m}^{t+1} = \begin{cases} 1 & \text{if the bit is just flipped} \\ \max\left(\text{stkns}_{i,m}^{t} - \dfrac{1}{\text{stknsS}}, 0\right) & \text{otherwise,} \end{cases} \tag{17}$$

where $\text{stkns}_{i,m}$ is the stickiness vector, and stknsS is the step size.

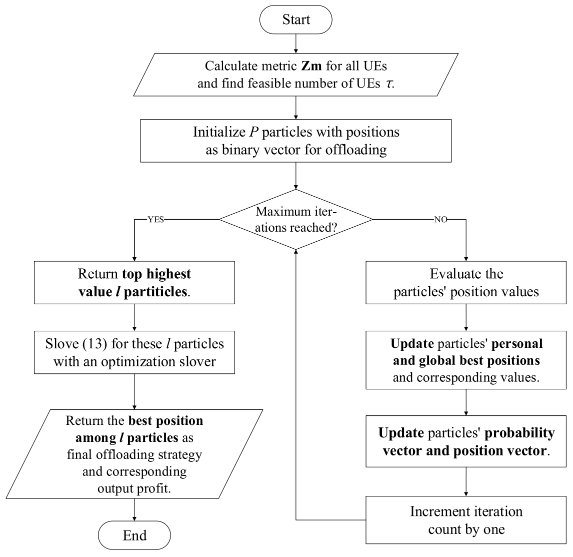
For SBPSO, a weight was defined that increases the probability of a bit flipping when the bit is different from the personal best and global best bit. Using the stickiness property instead of momentum, the probability that the m^{th} bit of the i^{th} particle is flipped is defined as

$$p_{i,m}^{t+1} = \mu_s(1 - \text{stkns}_{i,m}^{t}) + \mu_p|\text{pbest}_{i,m}^{t+1} - x_{i,m}^{t}| + \mu_g|\text{gbest}_m - x_{i,m}^{t}| \tag{18}$$

where μ_s is the importance of the stickiness factor, and μ_p and μ_g are the importance of the personal and global factors respectively. Based on the flipping probability (18), the bit value in the position vector of the i^{th} particle is updated at the $(t+1)^{\text{th}}$ iteration as follows:

$$x_{i,m}^{t+1} = \begin{cases} 1 - x_{i,m}^{t}, & \text{if rand()} \le p_{i,m}^{t+1} \\ x_{i,m}^{t}, & \text{otherwise.} \end{cases} \tag{19}$$

In the dynamic SBPSO case, the importance factors and stickiness step size are updated during every iteration



Start
Calculate metric Zm for all UEs and find feasible number of UEs τ.
Initialize P particles with positions as binary vector for offloading
Maximum iterations reached?
YES
NO
Return top highest value l particles.
Slove (13) for these l particles with an optimization slover
Return the best position among l particles as final offloading strategy and corresponding output profit.
End
Evaluate the particles' position values
Update particles' personal and global best positions and corresponding values.
Update particles' probability vector and position vector.
Increment iteration count by one

FIGURE 6. A flowchart for SBPSO.

according to equation (12) in [29], whereas for the static SBPSO case, they remain constant as in Table 2 in that study [29]. The overall process of the SBPSO-based solution of (13) can be summarized in the form of a flowchart, as shown in Figure 6 and Algorithm 2.

V. SIMULATION RESULTS AND DISCUSSION

The simulation scenario and parameters are similar to those in the aforementioned study [18]. The MEC coverage radius is set to 500 m. $N = 10$ RRHs and the number of UEs M varies from 30 to 110, being randomly distributed. The fronthaul link capacity of each RRH is set to 50 Mbps. The tasks are randomly generated. The range of the output data size D_m is between 50 to 200 KB. The CPU cycle requirement for a task is 1000 times its data size ($F_m = 1000D_m$). The latency requirement varies from 360 ms to 900 ms depending on the task. The channel bandwidth is set to 10 MHz and the transmit power is set to 30 dBm with an antenna gain of 10 dBi, and the path loss model is $37.6 \log(\text{dist}) + 148.1$. The shadowing factor is given by a log-normal function with standard deviation of 8 dB. The noise power is set to $\sigma^2 = -174$ dBm/Hz. Remaining parameters are as shown in Table 1.

A. PERFORMANCE COMPARISONS

The comparison of the six algorithms described in Sections III and IV, namely the simple genetic algorithm (SGA), the fast genetic algorithm (FGA), the modified SJOORA algorithm, the greedy algorithm, the static SBPSO and the dynamic SBPSO in terms of service provider's profitability is shown in Figure 7a. The simulation results are the average performance over 10 iterations. As shown in the figure, for all algorithms, when the number of UEs is initially low, the profits increase rapidly with the number of UEs. This is because with relatively small number of UEs, there are enough resources to allocate, so the profits increase

Algorithm 2 Sticky Binary Particle Swarm Optimization Algorithm (SBPSO)

```
Input: e M N W B F l P and other simulation parameters
Output: Profit, Offloading strategy A
 1: Create $\mathbf{e_{max}}$ of length $M$ where $\mathbf{e_{max}} = \max_n(\mathbf{e}), \forall m \in M$.
 2: Create $\mathbf{A}_{in}$ where $a_{m,n_{in}} = 1$ if $n = \arg\max_n \mathbf{e} \forall m \in M$; 0 otherwise .
 3: Calculate $Z_m$ for ranking all UEs.
 4: Create a list of $P$ particle objects as in section IV-B.
 5: for t ← 1 to $T$ do
 6:   Update importance factors and stickiness step size. {Only for the case of dynamic SBPSO (DSBPSO)}
 7:   for i ← 1 to $P$ do
 8:     particle(i).value ← CalculateProfit(particle(i).position)
 9:     if particle(i).value > particle(i).pbestvalue then
10:       particle(i).pbestvalue ← particle(i).value
11:       particle(i).pbest ← particle(i).position
12:       if particle(i).value > particle(i).gbestvalue then
13:         particle(:).gbestvalue ← particle(i).value
14:         particle(:).gbest ← particle(i).position
15:       end if
16:     end if
17:     for $m$ ← 1 to $M$ do
18:       Update particle(i).stkns(m)
19:       Update particle(i).probability(m)
20:       Update particle(i).position(m)
21:     end for
22:   end for
23: end for
24: Select the top $l$ particles with highest values. Calculate profits using the optimization solver for these $l$ particles' positions.
25: Let a_best be the most profitable particle out of these $l$ particles.
26: Final offloading strategy $\mathbf{A} = a\_best^T \odot \mathbf{A}_{in}$.
```

TABLE 1. Simulation parameters.

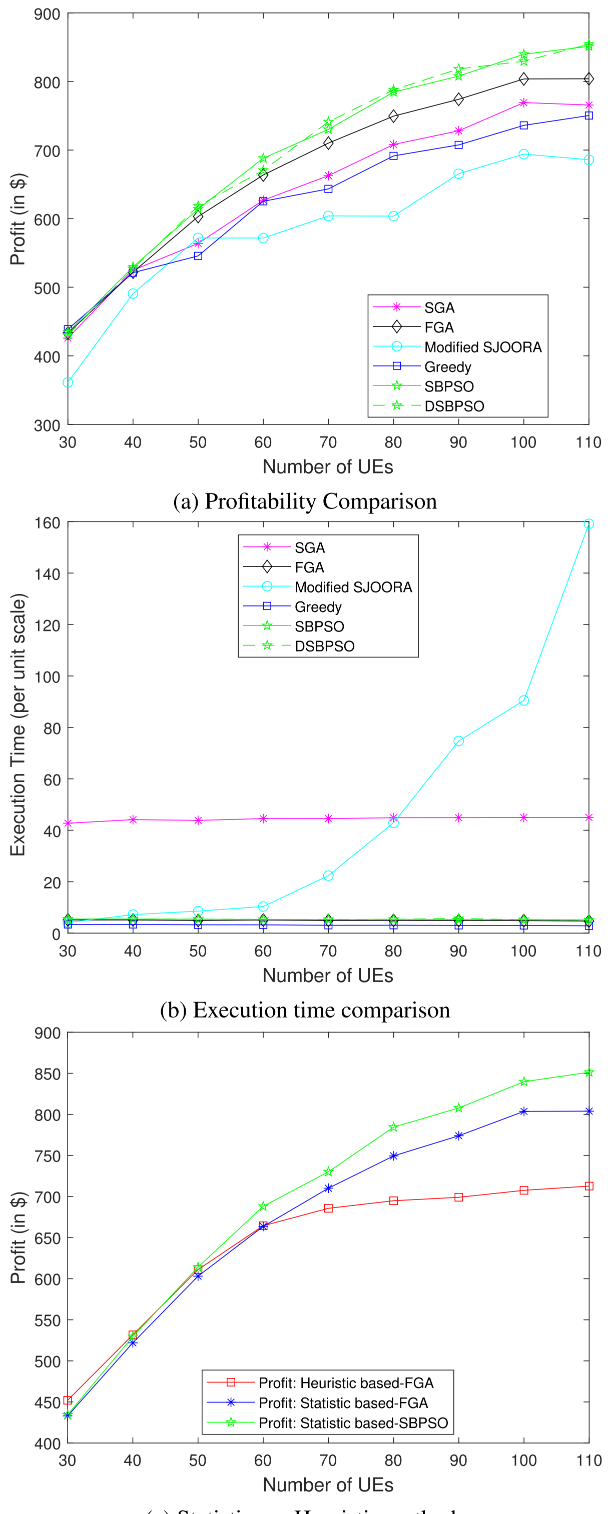
Parameter	Values
Impact factor of computation cost κ	0.5
Impact factor of bandwidth cost ω	10
Unit price of charge for computation p_f	0.03 \$/Mega cycle
Unit price of charge for transmission p_t	0.3 \$/Mbit
Unit cost of bandwidth q_b	0.5 \$/MHz
Unit cost of computation q_c	0.005 \$/Mega cycle/s
MEC server computation capacity F	100 GHz
Local computational capability f_m^{local}	0.7 GHz
NIND	20 (Simple), 30 (Fast)
GGAP	1.2
MUTR	0.05
MAXGEN	4 (Simple), 7(Fast)
mean_frac	0.8
$\theta_1, \theta_2, \theta_3$	100, 10, 10
ϕ_1, ϕ_2, ϕ_3	50, 30, 40

linearly with the number of UEs. However, as the number of UEs increases, the rate of profit growth begins to decrease, resulting in a saturated profit curve. This happens because as the number of UEs increases, the demand for offloading increases and the resources become scarcer. Therefore, more UEs requesting offloading does not result in a proportional increase in profits.

Among the six algorithms, the modified SJOORA shows the least profitability. The greedy algorithm outperforms the modified SJOORA, in which the profit from selecting a UE for offloading depends not only on the spectral efficiency of the UE-RRH link, but also on the Z_m (14), the rank assigned to UE task information W_m. By using a greedy approach to seed the initial population with the expected optimal solution, our proposed evolutionary methods explore solution space better as they evolve, finding more profitable offloading strategies and solutions that can yield higher profits. Therefore, the proposed genetic algorithms (SGA and FGA) and the swarm optimization algorithms (SBPSO and DSBPSO) show better performance. Despite using approximate method for the fitness calculation, FGA provides better profits than SGA in much shorter time. SBPSO and DSBPSO perform similarly, but DSBPSO has more variation because the parameter values are dynamically adjusted in each iteration. These two algorithms also outperform the genetic algorithms with similar execution times, demonstrating the effectiveness of the swarm-based optimization approach in the binary domain.

Figure 7b shows the execution times of the algorithms when run on MATLAB using a PC with 11-th Gen i7-11700, 2.5GHz processor and 16GB RAM. As shown in the figure, the modified SJOORA algorithm has comparable execution latency for relatively few UEs, but it becomes extremely slow as the number of UEs increases. This is expected, because the time complexity of the original SJOORA algorithm [18] depends linearly on the number of UEs. Other evolutionary algorithms have nearly constant execution time, meaning that the execution times of the algorithms are independent of the number of UEs. This is the main advantage of evolutionary algorithms, which can be used as scalable solution where the MEC system needs to accommodate a large number of UEs for computation offloading. The greedy algorithm is the fastest because it simply attempts five different possible offloading solutions and selects the best among them, and there is no additional evolutionary procedure to find the optimal solution. SGA is the slowest among the evolutionary algorithms as it uses the optimization solver to calculate the fitness value of each chromosome for every generation of evolution. FGA, SBPSO and DSBPSO, on the other hand, use approximate profit calculation methods for a given offloading decision, which results in much shorter execution times.

The execution time of a genetic algorithm is a function of the number of individuals in the population (NIND), the number of evolutionary generations (MAXGEN), the generation gap (GGAP) and the objective function for fitness evaluation (''CalculateProfit''). Since the parameters of the genetic algorithm, NIND, MAXGEN and GGAP, do not depend on the number of UEs, the execution time of the evolutionary algorithm remains nearly constant as the number of UEs increases. Similarly, the execution times of SBPSO and DSBPSO are dependent on the number of particles, the number of iterations T, the number of bits M in the position vector and the objective value evaluation function. Here, the effect of the number of particles and the number



(a) Profitability Comparison

(b) Execution time comparison

(c) Statistics vs Heuristic methods

FIGURE 7. **Performance comparison of the algorithms.**

of iterations on the execution time greatly overwhelms the effect of the number of bits in the position vector. Therefore, the execution time remains constant for both SBPSO and DSBPSO schemes and does not increase as the number of UEs increases.

In figure 7c, we compare the performance of the fast genetic algorithm FGA proposed in this paper, which uses a machine learning model for the resource allocation prediction in the profit calculation, to our previous work [24], which uses only heuristic values for this type of approximation. Clearly, the method based on statistics (machine learning) provides significantly better profits in a slightly shorter execution time than the method based on the heuristics.

B. IMPACT OF RESOURCE AVAILABILITY

Next, we evaluated the impact of resource availability on the profitability of the offloading scheme for the following three simulation parameters:

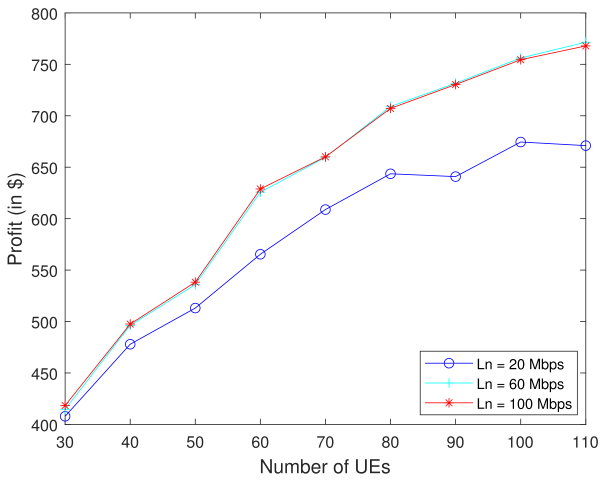
1) Fronthaul link capacity for each RRH (L_n)
2) Bandwidth per RRH (B)
3) Computing resources available on the MEC server (F).

We used the simple genetic algorithm (results averaged over 10 iterations) for this evaluation.

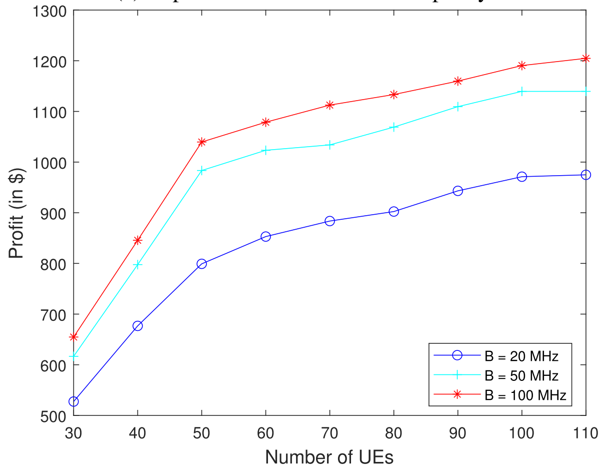
In Fig. 8, we compare the profitability for different levels of given resource availability, while keeping the other factors constant. A common trend in Figs. 8a, 8b, and 8c is that the profit increases as the amount of resource increases, but the proportion of the profit increase begins to decline as the resource availability increases further. This trend can be explained by the fact that as the availability of certain resource increases, the availability of other resources starts to function as a limiting constraint on the optimization problem, thus restricting the profit increase. For example, in Fig. 8a, we plot the Profit vs Number of UEs for different fronthaul link capacities, L_n, i.e., for 20 Mbps, 60 Mbps and 100 Mbps indicating that more fronthaul link capacity does not necessarily mean proportionately higher profit, as the profits at 60 Mbps are similar to those obtained with a link capacity of 100 Mbps. However, a very low fronthaul such as 20 Mbps can act as a limiting constraint on the optimization problem and reduce the expected profits. Similar observations can be made regarding the availability of bandwidth at RRH and the computational capacity of the MEC server.

C. IMPACT OF SIMULATION PARAMETERS ON EVOLUTIONARY ALGORITHMS

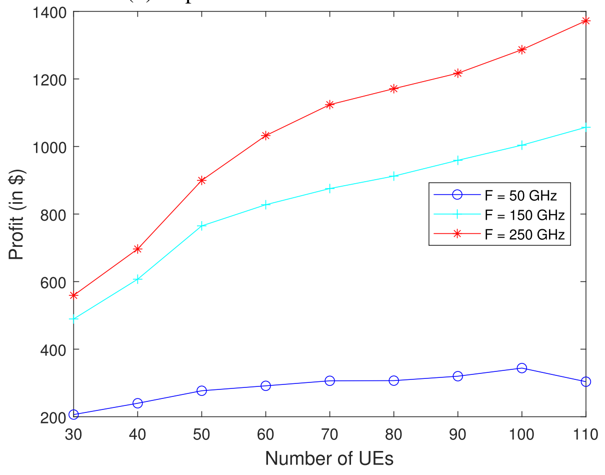
Since the execution time and the profitability of evolutionary algorithms depend on the number of chromosomes(NIND), particles(P), the number of generations(MAXGEN), and iterations(T), we compare the performances of FGA and SBPSO as the values of these parameters change. A common observation in Fig. 9 is that the profits increase as the parameter value increases, but the profit increase becomes marginal as the parameter values increase further. The execution time of the algorithm also increases proportionally



(a) Impact of the fronthaul link capacity



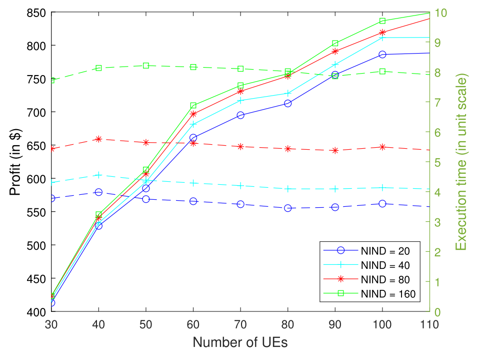
(b) Impact of the available bandwidth



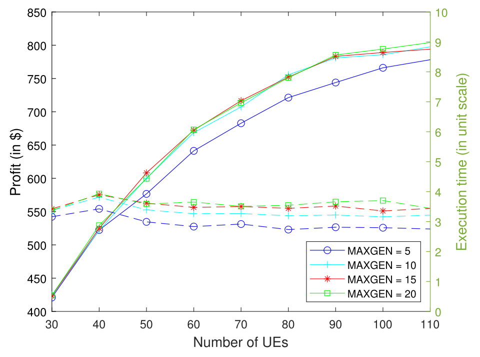
(c) MEC Computational Capacity impact

FIGURE 8. Impact of resource availability.

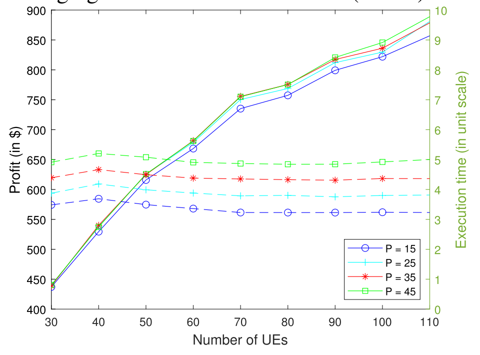
to the increase in parameter values. Therefore, setting the parameter value too high is undesirable because it increases the execution time of the algorithm by more than the slight increase in the profit value. Conversely, setting the value too low will reduce the proper exploration of the solution space, resulting in lower profitability.



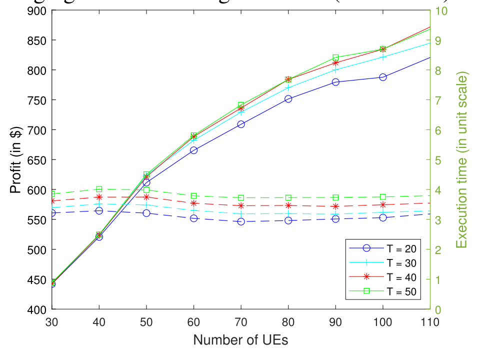
(a) Changing the number of individuals (NIND) in FGA



(b) Changing the number of generations (MAXGEN) in FGA



(c) Changing the number of particles (P) in SBPSO



(d) Changing the number of iterations (T) in SBPSO

FIGURE 9. Impact of evolutionary algorithms' parameters.

VI. CONCLUSION AND FUTURE WORKS

With limited battery and computing resources in UE terminals, computation offloading using MEC or cloud servers to avoid excessive battery consumption and reduce latency can be very useful from a user perspective, especially for latency-sensitive applications such as VR/AR and autonomous vehicles. Furthermore, for network service providers who have built their networks with huge capital expenditures (CAPEX), it is important to create business models to increase the utilization of MEC networks and monetize them.

In this paper, we examined how profitable it is to offload computation in C-RAN with MEC from the network operator's perspective. We formulated a profit maximization problem considering the joint optimization of offloading decisions along with radio and computing resource allocation. The optimization problem belongs to the class of mixed integer nonlinear programming problem. Therefore, we decoupled the offloading decision from the resource allocation problem. To arrive at the offloading decision, we used several nature-inspired evolutionary algorithms, specifically SGA, FGA, SBPSO, DSBPSO algorithms and compared them to spectral efficiency-based optimization algorithm (modified SJOORA). We compared several penalty functions for incorporating constraint violations in the profit calculation. In addition, for the faster execution of the evolutionary algorithms, we use a machine learning based regression model to predict the resource allocation values during the evolutionary cycle. To obtain a dataset for a statistical-based model, we initially start with a heuristic-based algorithm as in our previous work [24], and by generating a dataset using a simulator, we have more optimization-relevant data to switch to a statistical-based model.

We showed that the proposed algorithms (FGA, SBPSO) outperform the other algorithms in terms of profitability, while their execution time is lower and almost constant, meaning that it does not increase as the number of UEs increases, making them ideal as a scalable solution. In addition, the low execution time allows system to quickly decide whether to offload computation for latency-sensitive services. Finally, we evaluated how the profitability of offloading changes with resource availability and with changes of the parameters of evolutionary algorithms.

There has been active research on optimizing computation offloading in MEC using deep reinforcement learning (DRL) [23] and multi-agent reinforcement learning (MARL) [34], [35]. In our future works, we plan to consider new optimization modeling including objective function and constraints and conduct optimization methodology that combines the evolutionary algorithm with methods based on DRL or MARL.

REFERENCES

[1] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery-driven system design: A new frontier in low power design," in *Proc. ASP-DAC/VLSI Design, 7th Asia South Pacific Design Autom. Conf., 15h Int. Conf. VLSI Design*, Jan. 2002, pp. 261–267.

[2] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, Apr. 2010.

[3] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5G heterogeneous networks," *IEEE Signal Process. Mag.*, vol. 31, no. 6, pp. 45–55, Nov. 2014.

[4] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 4, pp. 23–32, Apr. 2013.

[5] M.-H. Chen, B. Liang, and M. Dong, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," in *Proc. IEEE 16th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jun. 2015, pp. 186–190.

[6] K. Wang, K. Yang, and C. S. Magurawalage, "Joint energy minimization and resource allocation in C-RAN with mobile cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 760–770, Jul. 2018.

[7] S. Zhou, M. Zhao, X. Xu, J. Wang, and Y. Yao, "Distributed wireless communication system: A new architecture for future public wireless access," *IEEE Commun. Mag.*, vol. 41, no. 3, pp. 108–113, Mar. 2003.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[9] Y. H. M. Patel. (Sep. 2014). *Mobile-Edge Computing—Introductory Technical White Paper*. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[10] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[11] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[12] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102781.

[13] H. Huang, K. Peng, and X. Xu, "Profit-driven computation offloading for mobile edge computing in wireless metropolitan area networks," in *Proc. IEEE 22nd Int. Conf. High Perform. Comput. Commun.; IEEE 18th Int. Conf. Smart City; IEEE 6th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2020, pp. 336–343.

[14] L. Pan, X. Liu, Z. Jia, J. Xu, and X. Li, "A multi-objective clustering evolutionary algorithm for multi-workflow computation offloading in mobile edge computing," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1334–1351, Apr./Jun. 2023, doi: 10.1109/TCC.2021.3132175.

[15] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1277–1287, Jul. 2021.

[16] E. Duman, M. Uysal, and A. F. Alkaya, "Migrating birds optimization: A new Metaheuristic approach and its performance on quadratic assignment problem," *Inf. Sci.*, vol. 217, pp. 65–77, Dec. 2012.

[17] Q. Fan, J. Lin, G. Feng, Z. Gao, H. Wang, and Y. Li, "Joint service caching and computation offloading to maximize system profits in mobile edge-cloud computing," in *Proc. 16th Int. Conf. Mobility, Sens. Netw. (MSN)*, Dec. 2020, pp. 244–251.

[18] Z. Jian, W. Muqing, and Z. Min, "Joint computation offloading and resource allocation in C-RAN with MEC based on spectrum efficiency," *IEEE Access*, vol. 7, pp. 79056–79068, 2019.

[19] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Germany: Springer-Verlag, 2003. [Online]. Available: https://books.google.co.in/books?id=7IOE5VIpFpwC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[20] M. Grant and S. Boyd. (Mar. 2014). *CVX: MATLAB Software for Disciplined Convex Programming, Version 2.1*. [Online]. Available: http://cvxr.com/cvx

[21] S. Singh. (Jan. 2022). *Profit_Optimization_MEC*. [Online]. Available: https://github.com/sumit521/Profit_Optimization_MEC

[22] A. F. Kuri-Morales and J. Gutiérrez-García, "Penalty function methods for constrained optimization with genetic algorithms: A statistical analysis," in *Proc. Mex. Int. Conf. Artif. Intell.* Yucatan, Mexico: Springer, 2002, pp. 108–117.

[23] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.

[24] S. Singh and D. Ho Kim, "Profit optimization for mobile edge computing using genetic algorithm," in *Proc. IEEE Region Symp. (TENSYMP)*, Aug. 2021, pp. 1–6.

[25] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz, "Penalty functions," *Handbook Evol. Comput.*, vol. 97, no. 1, p. C5, 1997.

[26] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw. (ICNN)*, vol. 4, Dec. 1995, pp. 1942–1948.

[27] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. Comput. Cybern. Simulation*, vol. 5, Oct. 1997, pp. 4104–4108.

[28] B. H. Nguyen, B. Xue, and P. Andreae, "A novel binary particle swarm optimization algorithm and its applications on knapsack and feature selection problems," in *Intelligent and Evolutionary Systems: The 20th Asia Pacific Symposium, IES 2016, Canberra, Australia, November 2016, Proceedings*. Springer, 2017, pp. 319–332.

[29] B. H. Nguyen, B. Xue, P. Andreae, and M. Zhang, "A new binary particle swarm optimization approach: Momentum and dynamic balance between exploration and exploitation," *IEEE Trans. Cybern.*, vol. 51, no. 2, pp. 589–603, Feb. 2021.

[30] L. Wang, X. Fu, Y. Mao, M. Ilyas Menhas, and M. Fei, "A novel modified binary differential evolution algorithm and its applications," *Neurocomputing*, vol. 98, pp. 55–75, Dec. 2012.

[31] M. J. Islam, X. Li, and Y. Mei, "A time-varying transfer function for balancing the exploration and exploitation ability of a binary PSO," *Appl. Soft Comput.*, vol. 59, pp. 182–196, Oct. 2017.

[32] J. Liu, Y. Mei, and X. Li, "An analysis of the inertia weight parameter for binary particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 666–681, Oct. 2016.

[33] Y.-W. Jeong, J.-B. Park, S.-H. Jang, and K. Y. Lee, "A new quantum-inspired binary PSO: Application to unit commitment problems for power systems," *IEEE Trans. Power Syst.*, vol. 25, no. 3, pp. 1486–1495, Aug. 2010.

[34] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9282–9293, Sep. 2021.

[35] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3425–3443, Jun. 2023.

SUMIT SINGH received the B.Tech. degree in electronics and communication engineering from IIT (ISM), Dhanbad, India, in 2014, and the M.S. degree in integrated IT engineering from the Seoul National University of Science and Technology, in 2022, under the APT-Republic of Korea Scholarship Program 2019.

He is currently the Assistant Director General with the Department of Telecommunications, Government of India. Previously, he was with the Satcomm and Cellular Communications Division, Bharat Electronics Ltd., Ghaziabad, India, from 2015 to 2017. Since then, he has been with the Department of Telecommunications, Government of India, New Delhi. His research interests include the application of mathematical optimization, game theory, and reinforcement learning in wireless communications.

DONG HO KIM (Senior Member, IEEE) received the B.S. degree from Yonsei University, in 1997, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea, in 1999 and 2004, respectively.

From 2004 to 2006, he was a Senior Member of Technical Staff, 4G Wireless Technology Laboratory, Samsung Advanced Institute of Technology (SAIT). From 2004 to 2006, he was also a Senior Researcher with the Mobile Communications Research Institute, Samsung Electronics. Since 2007, he has been a Professor with the Seoul National University of Science and Technology. His current research interests include the design of immersive media (VR/AR) and 5G/6G communication systems.