applied sciences

MDPI

Article

A Reinforcement Learning-Based Multi-Objective Bat Algorithm Applied to Edge Computing Task-Offloading Decision Making

Chwan-Lu Tseng *, Che-Shen Cheng and Yu-Hsuan Shen

Department of Electrical Engineering, National Taipei University of Technology, Taipei 10608, Taiwan; t103319001@ntut.edu.tw (C.-S.C.); t110318055@ntut.org.tw (Y.-H.S.)
* Correspondence: f10940@ntut.edu.tw

Abstract: Amid the escalating complexity of networks, wireless intelligent devices, constrained by energy and resources, bear the increasing burden of managing various tasks. The decision of whether to allocate tasks to edge servers or handle them locally on devices now significantly impacts network performance. This study focuses on optimizing task-offloading decisions to balance network latency and energy consumption. An advanced learning-based multi-objective bat algorithm, MOBA-CV-SARSA, tailored to the constraints of wireless devices, presents a promising solution for edge computing task offloading. Developed in C++, MOBA-CV-SARSA demonstrates significant improvements over NSGA-RL-CV and QLPSO-CV, enhancing hypervolume and diversity-metric indicators by 0.9%, 15.07%, 4.72%, and 0.1%, respectively. Remarkably, MOBA-CV-SARSA effectively reduces network energy consumption within acceptable latency thresholds. Moreover, integrating an automatic switching mechanism enables MOBA-CV-SARSA to accelerate convergence speed while conserving 150.825 W of energy, resulting in a substantial 20.24% reduction in overall network energy consumption.

Keywords: edge computing; energy consumption; latency; offloading decision; multi-objective optimization; reinforcement learning

Citation: Tseng, C.-L.; Cheng, C.-S.; Shen, Y.-H. A Reinforcement Learning-Based Multi-Objective Bat Algorithm Applied to Edge Computing Task-Offloading Decision Making. *Appl. Sci.* **2024**, *14*, 5088. https://doi.org/10.3390/app14125088

Academic Editors: Mehdi Sookhak and Francesco Moscato

Received: 29 April 2024
Revised: 4 June 2024
Accepted: 6 June 2024
Published: 11 June 2024

1. Introduction

As the prevalence of smart devices, the rapid expansion of the Internet of Things (IoT), and the rise of 5G networks continue, increasingly complex and computationally demanding applications are becoming deeply integrated into our lives. While device processing capabilities have improved, there remains a significant gap compared to the resource demands of applications. To address these demands, data must be transmitted to the cloud for computation, leveraging its infinite computing power. However, this approach is constrained by limited bandwidth between devices and the cloud, exacerbated by the growing data volume due to application complexity, leading to noticeable increases in transmission latency. Thus, scholars propose deploying edge servers near smart devices or IoT devices for task computation, known as mobile edge computing (MEC). While MEC significantly reduces transmission-induced latency, latency between smart devices and edge servers remains a notable issue. However, executing all tasks locally on devices can reduce transmission-induced latency but may excessively deplete the limited energy of wireless devices, leading to device failure. Therefore, effectively allocating tasks to balance latency and energy consumption has become a crucial research problem [1].

Cui et al. [2] established four models for computing energy consumption and latency in edge computing task offloading. To strike a balance between energy consumption and latency, researchers have formalized the problem as a multi-objective optimization challenge, employing an enhanced non-dominated sorting genetic algorithm (NSGA-II) to obtain the optimal solution. Bozorgchenani et al. [3] devised a computational sharing system model, offloading tasks to nearby smart devices or edge servers for computation. They proposed a multi-objective evolutionary algorithm (MOEA) to achieve the optimal trade-off between

energy consumption and task processing delay, further optimizing the offloading quantity with NSGA-II to prolong network lifespan. Gedawy et al. [4] also introduced RAMOS, which efficiently schedules tasks to edge devices by utilizing idle computing cycles in heterogeneous mobile IoT devices. The scheduling operates in two modes: latency minimization maximizes system throughput while adhering to energy constraints, and the energy-saving mode aims to minimize total energy consumption while meeting task deadlines. Alfakih et al. [5] presented MOAPSO-DP, employing a non-preemptive priority algorithm, APSO, and dynamic programming to schedule tasks in edge computing to virtual machines on edge servers, significantly reducing computation time.

In summary, latency and energy consumption are focal points in edge computing. However, prioritizing latency often leads to increased energy consumption. Many existing strategies optimize for a single objective and lack careful consideration of multiple goals, necessitating multi-objective optimization to find the best compromise between latency and energy consumption.

Xiao et al. [6] proposed an advanced binary particle swarm optimization algorithm to refine content caching strategies in multi-objective optimization. Additionally, they employed a multi-objective bat algorithm to optimize task-offloading decisions, aiming for reduced latency and energy consumption. They integrated weighting to combine multiple objective functions, transforming the multi-objective problem into a single-objective one for streamlined optimization. However, this conversion to a single objective via weighting may result in unequal importance among objectives. In contrast, Mohan et al.'s ENsdBA method [7] enhanced the bat algorithm by integrating non-dominated sorting and crowding distance. While retaining all new solutions to prevent algorithm stagnation, this method might retain suboptimal solutions. The bat algorithm (BA), compared to genetic algorithms (GA) and particle swarm optimization (PSO), has demonstrated itself as a more contemporary, stable, and rapidly converging method [8]. Therefore, this study preserves the complete bat algorithm process, augmenting it with non-dominated sorting and crowding distance calculations from NSGA-II, establishing the multi-objective bat algorithm (MOBA) to provide comprehensive optimization strategies.

In algorithmic frameworks, the selection of hyperparameters profoundly impacts performance. Discovering suitable hyperparameters necessitates numerous experiments, which prove time-consuming and intricate and do not assure optimal findings.

This paper addresses the critical role of hyperparameter selection in algorithmic performance and introduces reinforcement learning to enhance parameter tuning in multi-objective optimization algorithms. It discusses the integration of NSGA-II and SARSA into NSGA-RL by Kaur et al. [9] and the use of Q-learning by Liu et al. [10] to refine the parameters for MOPSO. The comparison highlights the risk differences between Q-learning and SARSA, emphasizing SARSA's lower risk and superior learning effectiveness [11].

This paper proposes applying an enhanced-learning-based multi-objective bat algorithm to the task-offloading decision-making process in edge computing, addressing the trade-off between latency and energy consumption. The distinctive features of this study can be summarized as follows:

(1) This study optimizes edge computing task-offloading decisions to address network latency and energy consumption issues, aiming to prolong the lifespan of smart devices.
(2) Retaining the bat algorithm's (BA) single-objective optimization process, we introduce multi-objective optimization to create the multi-objective bat algorithm (MOBA). This approach proposes multiple objective decisions to select candidates from intricate solution variances, fostering evolutionary progress.
(3) Enhanced-learning SARSA is integrated to automatically adjust the hyperparameters of the multi-objective bat algorithm, simplifying complex hyperparameter experiments and statistical analyses. Moreover, we introduce an automated mechanism to adjust the search scope, enhancing overall performance.
(4) Customized multi-objective functions are devised within the edge computing framework, considering intelligent mobile devices' remaining energy and computational

capacities. A custom-enhanced learning reward function is also proposed based on the edge computing objective function.

2. System Architecture and Problem Formulation

In this section, we will present an overview of edge computing architecture. Additionally, we will define the pertinent functions of local computation and edge computing separately. Lastly, this paper will introduce MOBA-CV-SARSA, as proposed herein.

2.1. System Architecture

In edge computing, computational resources move closer to smart devices, transitioning from the cloud to local proximity, as depicted in Figure 1. Edge servers are strategically placed at the network periphery. This shift eliminates the need for devices to send data to the cloud for processing, thereby economizing transmission time and circumventing bandwidth limitations. The strategic deployment of compact base stations enhances connectivity for myriad smart devices. Although edge computing reduces latency by minimizing transmission distances, the finite energy of wireless devices and transmission lags between devices and edge servers necessitate careful consideration. Consequently, this study will delve into the intricacies of edge computing's local architecture and extensively analyze these pertinent issues.

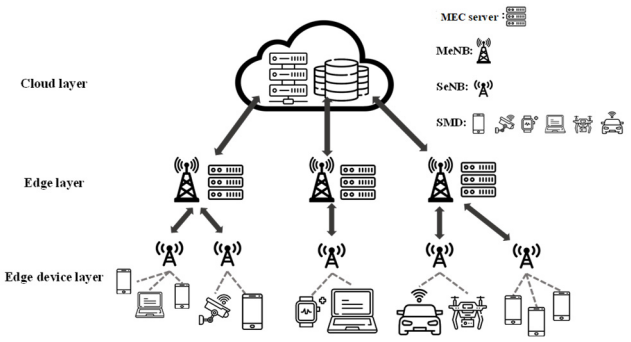

Figure 1. Edge computing network architecture diagram.

This study adopts the framework proposed in reference [2], which includes mobile edge computing (MEC) servers and Macro eNodeBs (MeNBs). MeNBs connect to multiple Small eNodeBs (SeNBs), as illustrated in Figure 2. SeNBs are divided and interconnected based on the location of Smart Mobile Devices (SMDs), facilitating task transmission. In this architecture, computational capabilities are exclusive to edge servers and smart devices. The configuration comprises one edge server, MeNB collectively, M SeNBs, and S SMDs.

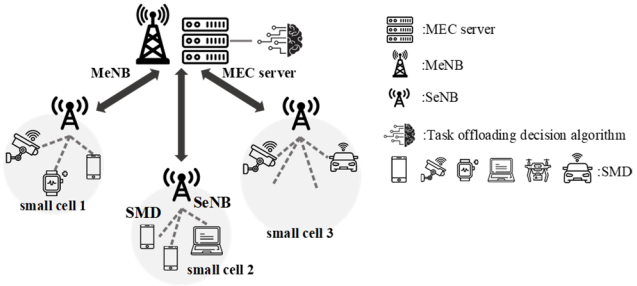

Figure 2. Partial edge computing network architecture diagram.

In this architecture, wired connections enable continuous power and data transmission between Macro eNodeBs and Small eNodeBs, simplifying transmission time and energy consumption considerations. Wireless links connect Small eNodeBs with smart devices, which lack continuous power and necessitate attention to residual energy and limited com-

putational capabilities. Task states remain until offloading decision optimization, allowing focused latency and energy consumption resolution. The edge computing server gathers optimization information from smart devices, including task sizes and computational capabilities. Considerations of network latency and energy consumption ensure balanced offloading decisions.

The offloading decision for the k-th task from the j-th smart device under the m-th MeNB is represented as $O_{m,j,k}$. A task is processed remotely if $O_{m,j,k}$ equals 1; otherwise, if the task chooses local processing, $O_{m,j,k}$ equals 0. Since a single smart device can generate multiple tasks, Equation (1) is applied to convert the task count into binary form [2]. This method converts the number of tasks (k_s) from smart device (s), where $s \in \{1, 2, \ldots, S\}$, and S denotes the total number of smart devices in the system. The bit length is k_s. By using the conversion formula, decimal values are transformed into binary format, enabling the assignment of offloading decisions to each task independently. This allows the determination of whether each task should be processed remotely or locally, as depicted in Figure 3.

$$B_s = 2^{k_s} - 1, \tag{1}$$

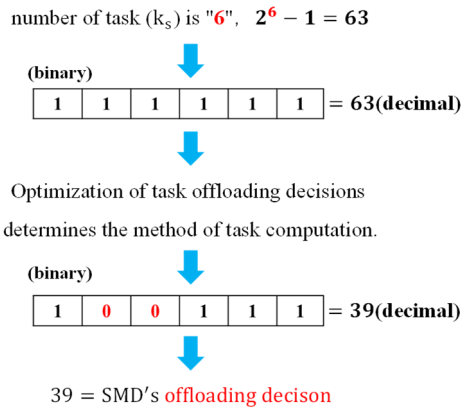

Figure 3. Offload decision coding.

2.2. Problem Formulation

Given the inherent trade-off between energy consumption and latency, this study delves into the network's energy consumption and latency dynamics by analyzing task-offloading strategies. It aims to discern whether individual tasks best suit local or remote computation. The ensuing discourse will explore the formulas quantifying energy consumption and latency for local and remote computation scenarios. These formulas will be tailored to the architecture of the j-th smart device within the m-th zone, as depicted in Figure 4, defining pertinent parameters.

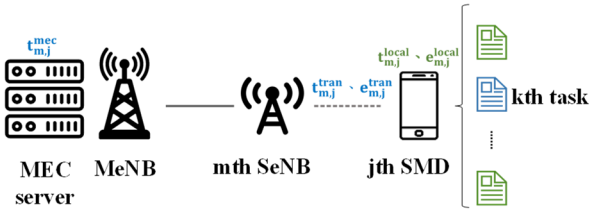

Figure 4. Framework for task transmission and definition of associated variables for an SMD.

First, we define the delay time ($t^{local}_{m,j}$) and energy consumption ($e^{local}_{m,j}$) when the j-th smart device in the m-th zone selects local computation. The delay time is the total computational capacity required by all tasks generated by the device, divided by the device's computational capacity ($f_{m,j}$) [12]. $k_{m,j}$ represents the total number of tasks generated by the device. In contrast, $c_{m,j,k}$ represents the computational capacity required to complete

the k-th task, namely the CPU cycles needed to complete the task. Equation (2) describes the required computation delay to the j-th smart device in the m-th zone:

$$t_{m,j}^{local} = \frac{\sum_{k=1}^{k_{m,j}} (1 - O_{m,j,k}) c_{m,j,k}}{f_{m,j}}, \tag{2}$$

Through the computation of the CPU cycles necessary for processing a 1-bit task ($CPU^{c_{m,j,k}}$) and the data size associated with it ($d_{m,j,k}$), we can ascertain the total CPU cycles needed to accomplish the task [12]:

$$c_{m,j,k} = d_{m,j,k} \times CPU^{c_{m,j,k}}, \tag{3}$$

The energy consumption ($e_{m,j}^{local}$) incurred by local computation results from the multiplication of the computational capacity needed for each task ($c_{m,j,k}$) by its respective offloading decision ($O_{m,j,k}$). Subsequently, this value is further multiplied by the energy consumption coefficient ($\varepsilon_{m,j}$) and the square of the smart device's computational capability ($f_{m,j}$) [12].

$$e_{m,j}^{local} = \varepsilon_{m,j} f_{m,j}^2 \left(\sum_{k=1}^{k_{m,j}} (1 - O_{m,j,k}) c_{m,j,k} \right), \tag{4}$$

In the upcoming discussion, we will explore the latency and energy consumption associated with remote computation. When the j-th smart device in the m-th zone engages in remote computation for the kth task, it is crucial to factor in the energy and time needed for task transmission from the device to the small base station and, subsequently, from the small base station to the edge server. Also, this study excludes consideration of the energy consumption and latency of the returned results, given their relatively small data volume and minimal impact on the overall network performance.

The forthcoming discussion will outline the transmission time ($t_{m,j}^{tran}$) and energy consumption ($e_{m,j}^{tran}$) associated with smart devices transferring tasks to small base stations. The transmission time hinges on the aggregate data size for transmission ($D_{m,j}$), regardless of the transmission rate from the smart device back to the small base station ($R_{m,j}$) [2]. Within this context, $d_{m,j,k}$ denotes the data size of the kth task generated by the jth smart device in the mth zone, $O_{m,j,k}$ signifies the offloading decision for that specific task, and $k_{m,j}$ represents the total number of tasks generated by the device.

$$t_{m,j}^{tran} = \frac{D_{m,j}}{R_{m,j}} = \frac{\sum_{k=1}^{k_{m,j}} O_{m,j,k} d_{m,j,k}}{R_{m,j}}, \tag{5}$$

In the above equation, the transmission rate ($R_{m,j}$) is determined by the Shannon Theorem, with B indicating the channel bandwidth, $p_{m,j}$ representing the device's transmission power, and $G_{m,j}$ indicating the channel gain. σ^2 stands for noise, while $I_{m,j}$ denotes the interference on this channel. $R_{m,j}$ is defined by Equation (6).

$$R_{m,j} = B \log_2 \left(1 + \frac{p_{m,j} G_{m,j}}{\sigma^2 + I_{m,j}} \right), \tag{6}$$

The channel gain ($G_{m,j}$) from the device to the small base station equals the gain (g_0) divided by the square of the distance between the location (x_{SMD}, y_{SMD}) of the smart mobile device (SMD) and the location of the small base station (SeNB) (x_{SeNB}, y_{SeNB}) plus the square of the base station's fixed height (H) [13]. In this context, g_0 denotes the channel gain at a reference distance of 1 m and a transmission power of 1 watt:

$$G_{m,j} = \frac{g_0}{d} = \frac{g_0}{(x_{SMD} - x_{SeNB})^2 + (y_{SMD} - y_{SeNB})^2 + H^2}, \tag{7}$$

The product of the device's transmission power ($p_{l,i}$) and the channel gain ($G_{m,l,i}$) from the connected small base station determines the total devices ($U_{l,i}$) in adjacent cells employing an identical transmission channel to the smart device ($U_{m,j}$), leading to the evaluation of channel interference ($I_{m,j}$). The channel decision ($a_{l,i,m,j}$) depends on whether devices ($U_{l,i}$) in neighboring cells utilize the same transmission channel as the device ($U_{m,j}$); a value of 1 signifies channel congruence, whereas 0 denotes disparate channels.

$$I_{m,j} = \sum_{l=1,l\neq m}^{M} \sum_{i=1}^{U_l} a_{l,i,m,j} p_{l,i} G_{m,l,i}, \tag{8}$$

We multiply the aggregate data size of transmitted tasks ($D_{m,j}$) by the transmission power ($p_{m,j}$) of the device and subsequently divide by the transmission rate ($R_{m,j}$) from the device to the small base station [2]. The energy consumption ($e_{m,j}^{tran}$) follows the equation

$$e_{m,j}^{tran} = p_{m,j} \frac{D_{m,j}}{R_{m,j}} = \frac{p_{m,j} \sum_{k=1}^{k_{m,j}} O_{m,j,k} d_{m,j,k}}{R_{m,j}}, \tag{9}$$

The total computational capacity needed for all remote computing tasks is calculated by multiplying the computational capacity required for each task ($c_{m,j,k}$) by the offloading decision ($O_{m,j,k}$) and summing the results. Subsequently, this total is divided by the computational capacity of the edge server (F):

$$t_{m,j}^{mec} = \frac{\sum_{k=1}^{k_{m,j}} O_{m,j,k} c_{m,j,k}}{F}, \tag{10}$$

The total network latency (T) encompasses the duration of local computation ($t_{m,j}^{local}$) for all smart device tasks, the transmission time for remote computation ($t_{m,j}^{tran}$), and the task computation time at the edge server ($t_{m,j}^{mec}$):

$$T = \sum_{m=1}^{M} \sum_{j=1}^{U_m} \left(t_{m,j}^{local} + t_{m,j}^{tran} + t_{m,j}^{mec} \right), \tag{11}$$

The total energy consumption (E) comprises the energy used for local computation ($e_{m,j}^{local}$) and the transmission energy for remote computation ($e_{m,j}^{tran}$) of all smart device tasks.

$$E = \sum_{m=1}^{M} \sum_{j=1}^{U_m} \left(e_{m,j}^{local} + e_{m,j}^{tran} \right), \tag{12}$$

This study considers three constraint violations (CVs): the combined energy of local computation ($t_{m,j}^{local}$) and remote computation transmission ($t_{m,j}^{tran}$) must not exceed the device's remaining energy ($e_{m,j}$).

$$e_{m,j}^{local} + e_{m,j}^{tran} > e_{m,j}, \tag{13}$$

Smart devices must possess computational capabilities that surpass the requirements for local computations.

$$f_{m,j} > \sum_{k=1}^{k_{m,j}} (1 - O_{m,j,k}) c_{m,j,k}, \tag{14}$$

Local tasks have an offloading decision of 0, while remote tasks have a decision of 1:

$$O_{m,j,k} \in \{0, 1\}, \tag{15}$$

Integration is performed using Equations (13)–(15). Optimized task-offloading decisions ($O_{m,j,k}$) are determined to minimize both overall network latency (T) and energy consumption (E) while adhering to the constraints:

$$\min_{O_{m,j,k}} \{T, E\},$$

$$\text{s.t. CV1: } e_{m,j}^{local} + e_{m,j}^{tran} > e_{m,j},$$

$$\text{CV2: } f_{m,j} > \sum_{k=1}^{k_{m,j}} (1 - O_{m,j,k}) c_{m,j,k},$$

$$\text{CV3: } O_{m,j,k} \in \{0, 1\},$$

2.3. Design of MOBA-CV-SARSA

Incorporating the multi-objective bat algorithm into the adaptive tuning of hyperparameters for the SARSA learning algorithm and considering the constraints of edge computing (CVs), we introduce MOBA-CV-SARSA, depicted in Figure 5, where A, B, and C are non-dominated solutions in the multi-object optimization; D is a dominated solution.

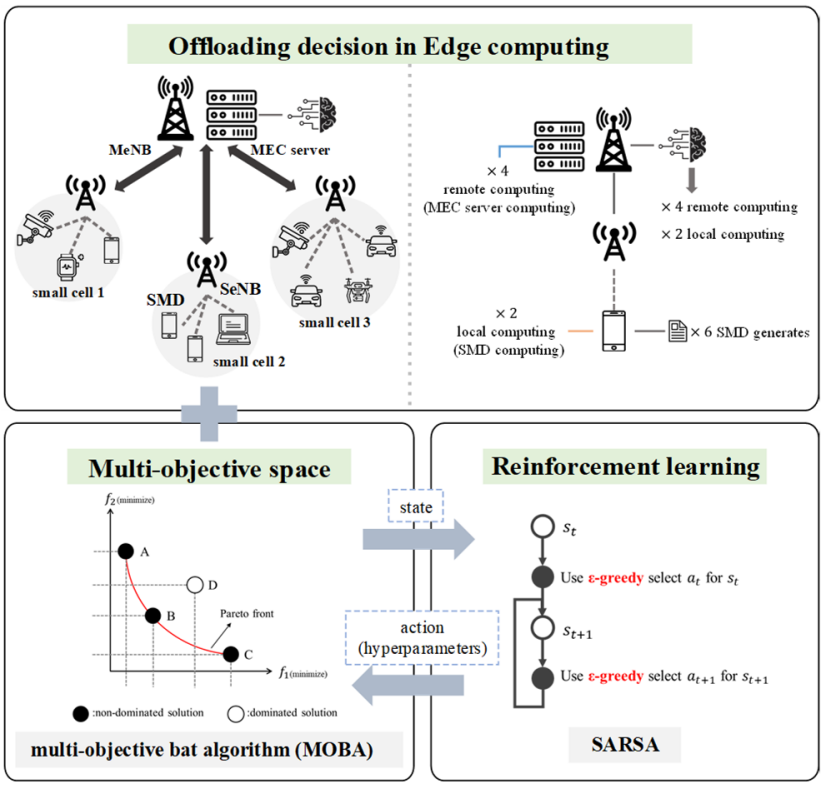

Figure 5. MOBA-CV-SARA's overall structure.

This study introduces MOBA-CV-SARSA, aiming to optimize task-offloading decisions for achieving equilibrium between network latency and energy consumption, as depicted in Figure 6. It preserves the bat algorithm's framework while elevating it to a multi-objective bat algorithm using non-dominated sorting and crowding distance. Additionally, it incorporates SARSA with reinforcement learning to autonomously fine-tune the multi-objective bat algorithm's hyperparameters and offers multi-objective decision making for thorough solution comparisons. This decision will evaluate the bat population's rank first and then the crowding distance [14] if the bats have the same rank in each iteration. Post-iteration sorting is conducted based on energy consumption to extend the network's longevity. Bats with smaller energy expenditures will be prioritized. This is because this paper hopes to extend the life of the network and does not want the device to fail prematurely due to insufficient power. Therefore, this decision is included before outputting the final result. That is, if bats at the same level have the same crowding distance at the same time, they will be sorted according to energy consumption in the final optimized population.

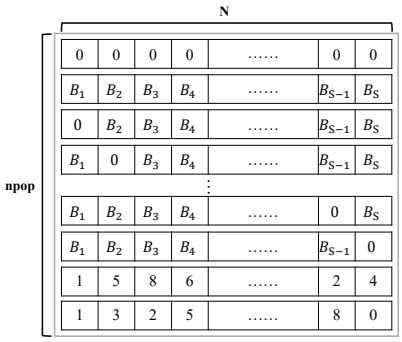

Figure 6. Initialization of bat group.

Initially, Figure 6 delineates the population's initialization settings [2]. Each row symbolizes a bat ($X_{i_{bats}}$,$X_{i_{bats}} = \{x_1, x_2, \ldots, x_N\}$), with N indicating the bat's dimensionality, equivalent to the overall count of smart devices (S). The total population consists of npop rows, signifying the aggregate number of bats. Each bat encapsulates task-offloading decisions across dimensions. The first bat performs local computation for all device tasks, whereas the second conducts remote computation. B_s reflects the outcome following the conversion of all functions of the sth device to remote computation, as defined by Equation (1).

In each iteration, the loudness (A_i) and pulse emission rate (r_i) of the bats undergo initial updates. According to Reference [15], optimal performance occurs when the loudness value (α) and pulse emission rate value (γ) fall within the range of 0.9 to 0.98. This study conducts a brute force experiment with 8250 trials and statistical analyses to determine the optimal values for MOBA-CV by combining and examining α and γ. Multiple experiments are required to obtain suitable α and γ hyperparameters for loudness and pulse emission rate. Hence, this paper proposes a multi-objective bat algorithm (MOBA-CV-SARSA) that employs reinforcement learning SARSA to streamline the cumbersome parameter experimentation process. SARSA treats the α and γ values used for bat updates as actions, where actions taken under specific states yield corresponding rewards. The subsequent section will present the customized reward function.

The primary objective of this research is to minimize both the total delay time (13) (denoted by f_1) and energy consumption (14) (denoted by f_2). However, SARSA pursues the maximization of the reward value. To prevent infinite values as the objective function becomes zero, a small positive value (e_1, e_2) is inclusively added to each objective function. Subsequently, these reciprocated values undergo multiplication with corresponding weights (ω_1, ω_2). Considering the absence of units, the energy consumption value (f_2) generally exceeds the delay time value (f_1) threefold. Hence, the reciprocal value of delay time ($\frac{1}{f_1}$) will exceptionally be that of energy consumption ($\frac{1}{f_2}$). To uphold equal significance for both objectives, ω_1 is designated as 0.25 and ω_2 as 0.75.

$$\text{reward} = \omega_1 \frac{1}{f_1 + e_1} + \omega_2 \frac{1}{f_2 + e_2}, \tag{16}$$

Reward values (16) enable the utilization of the Q-value update formula to modify the Q-table within SARSA. The Q-table format employed in this investigation is depicted in Table 1. Initially, bats exhibiting diverse states are integrated into the Q-table's state table. Following this, α and γ are designated as numerous sets of values, initializing all Q-values within the Q-table to 0. The respective Q-value undergoes an update upon the bat colony's state alignment with an entry in the Q-table. With each iteration, new bats emerge, prompting a comparison between these bats and the states archived in the Q-table. Should the new state be pre-existing, the associated state table is employed for Q-value ($Q_{i,j}$) updates. Conversely, if the new state is absent from the Q-table, the new bat is appended, and ε-greedy facilitates the initial action selection and Q-value update.

Table 1. Q-table for MOBA-CV-SARSA.

Q-Table					
State	**Action**				
	(α_1,γ_1)	(α_2,γ_2)	$\cdots$	$(\alpha_{k-1},\gamma_{k-1})$	(α_k,γ_k)
$X_{1_{bats}} = \{x_1, x_2, \cdots, x_N,\}$	$Q_{1,1}$	$Q_{1,2}$	$Q_{i,j}$	$Q_{1,k-1}$	$Q_{1,k}$
$X_{2_{bats}} = \{x_1, x_2, \cdots, x_N,\}$	$Q_{2,1}$	$Q_{2,2}$	$Q_{1i,j}$	$Q_{2,k-1}$	$Q_{2,k}$
$\vdots$			$\vdots$		
$X_{npop_{bats}} = \{x_1, x_2, \cdots, x_N,\}$	$Q_{npop,1}$	$Q_{npop,2}$	$Q_{npop,j}$	$Q_{npop,k-1}$	$Q_{npop,k}$

When dealing with a multi-objective problem, a singular objective assessment approach is inadequate. This study presents a specialized method for multi-objective decision making to evaluate multiple objectives simultaneously. Within this methodology, a comprehensive comparison between the new bat ($X_{i_{new\_bats}}$) and the optimal bat ($X_{1_{bats}}$) is conducted. The procedural details are elucidated in Algorithm 1, where 'M' denotes the total number of objective functions. In this investigation, M is set at 2, representing overall network latency and energy consumption. Initially, each objective function undergoes a comparison between the new bat and the optimal bat. If the new bat demonstrates superior performance over the optimal bat in the assessed objective function (f_j), the count of dominated objective functions (d_{num}) increases. When the number of objective functions dominated by the new bat matches the total count of objective functions, it signifies complete superiority over the optimal bat, prompting the replacement of the original bat with the new one. In sum, the pseudo-code of the aforementioned MOBA-CV-SARS is shown in Algorithm 1.

Algorithm 1: Pseudo-code of MOBA-CV-SARSA algorithm.

Initial bats' solutions/locations
$X_{i_{bats}} = (x_1, x_2, \ldots, x_N)$, velocities (v_i), frequencies (f_i), loudness (A_i), pulse rates (r_i);
$i = \{1, 2, \ldots,$ npop (number of bat′s population)$\}$, N is dimension of $X_{i_{bats}}$, rand $\in [0, 1]$
Calculate the multi-objective functions value $f = (f_1, f_2, \ldots, f_M)$ for $X_{i_{bats}}$, M is number of multi-objective functions
Process non-dominated sorting (based on ranking and crowding distance) for $X_{i_{bats}}$
for t = 1 to T (number of iterations)
Update $A\_i\hat{}(t + 1)$ and $r\_i\hat{}(t + 1)$ with SARSA
for i = 1 to npop
Select the best solution/location ($X_{1_{bats}}$)
Adjust frequencies (f_i) and update velocities (v_i), and generate solutions/locations ($X_{i_{new\_bats}}$)
if rand > pulse rate (r_i) then
Generate a local solution/location around the local best solution/location
end
Calculate the multi-objective functions value $f = (f_1, f_2, \ldots, f_M)$ for $X_{i_{new\_bats}}$
if rand < loudness (A_i) then
Process **multi-objective decision** to decide whether $X_{i_{new\_bats}}$ replaces $X_{i_{bats}}$
end
end
Concatenate $X_{i_{bats}}$ and $X_{i_{new\_bats}}$
Process non-dominated sorting of $X_{i_{bats}}$ and $X_{i_{new\_bats}}$
Select bats' solutions/locations ($X_{i_{bats}}$) for next iteration
end
if bats' solutions/locations ($X_{i_{bats}}$) have the same rank and crowding distance then
Sort in ascending order according to energy consumption
end
Display the final results

The proposed SARSA reinforcement learning algorithm optimizes hyperparameters α and γ in a range between 0 and 1. However, to accelerate the convergence, we firstly find the optimal α^* and γ^* obtained from MOBA-CV-SARSA with a broad search step. Then, we refine the best solutions using Formulas (17) and (18), creating narrower search intervals with finer steps.

$$\alpha \text{ range (narrow search)} = [\lfloor \alpha^* \rfloor, \lfloor \alpha^* \rfloor + 0.1], \tag{17}$$

$$\gamma \text{ range (narrow search)} = [\lfloor \gamma^* \rfloor, \lfloor \gamma^* \rfloor + 0.1], \tag{18}$$

Once the identified α^* and γ^* values converge to the specific values which are most frequently found, the search range for α and γ values contracts, triggering automatic switching, as depicted in Figure 7. By executing a fresh search and update within this narrowed range, the method not only hastens convergence but also determines more precise α and γ values. It balances network latency and energy consumption by optimizing task-offloading decisions. This research employs a multi-objective bat algorithm to negotiate solutions in multi-objective functions. It employs SARSA reinforcement learning to adjust hyperparameters dynamically, eliminating the necessity for intricate hyperparameter experiments. The enhanced MOBA-CV-SARSA-DA can accurately identify suitable parameter ranges.

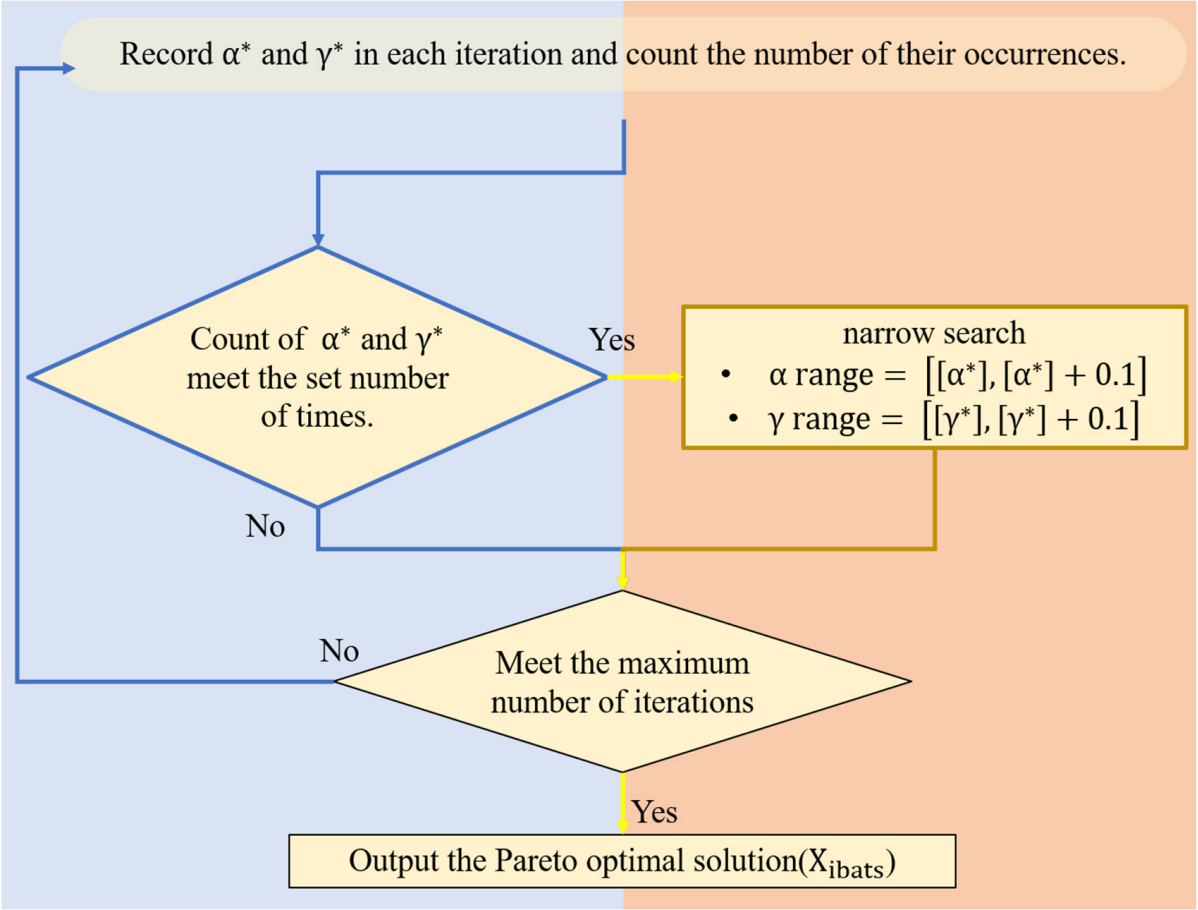

Figure 7. Flow chart of MOBA-CV-SARSA.

3. Experimental Results and Analysis

The forthcoming analysis will examine the outcomes derived from utilizing the multi-objective bat algorithm with SARSA reinforcement learning (MOBA-CV-SARSA) as presented in this study. Assessment and scrutiny of the conclusive Pareto solution sets produced by individual methodologies are carried out in this section.

3.1. Experimental Procedure

This study conducts comparative experiments on three sets of Pareto solutions. In the initial experiment, edge computing constraints (CV) are integrated into the multi-objective bat algorithm (MOBA) alongside NSGA-II [2] and MOPSO [16]. The second experiment introduces an enhanced-learning method for hyperparameter adjustment alongside constraints. The efficacy of the proposed MOBA-CV-SARSA is compared with NSGA-RL [9] and QLPSO [10] under the same initial conditions.

The proposed method is implemented by using C++ 23 software, and the resulting offloading decisions are validated by utilizing ns-3, employing a 5G millimeter-wave (mmWave) network architecture [17].

3.2. Evaluation Metrics

In order to conduct a detailed evaluation of the pareto optimal solution set found by each comparative method, the hypervolume index [18] is used to measure the dominance space of the solution, and the diversity metric [19,20] is used to evaluate the diversity of solutions.

3.2.1. Hypervolume

The main objective is to measure the extent of the solution set's dominance over the target space. A higher hypervolume value for the solution set suggests broader dominance within the target space, indicating a superior distribution. Computing the hypervolume entails establishing a reference point (r). The values of this reference point must surpass those of all solutions across each dimension in the non-dominated solution set under evaluation. Once established, the area between all solutions and the reference point can be determined, as depicted in Figure 8, where A–C are non-dominated solutions. In this research, the reference point (r) is defined as (200, 700), denoting that the delay time and energy consumption of all solutions derived from the methods are below 200 s and 700 W, respectively. Consequently, the area between each solution set and the reference point is computed.

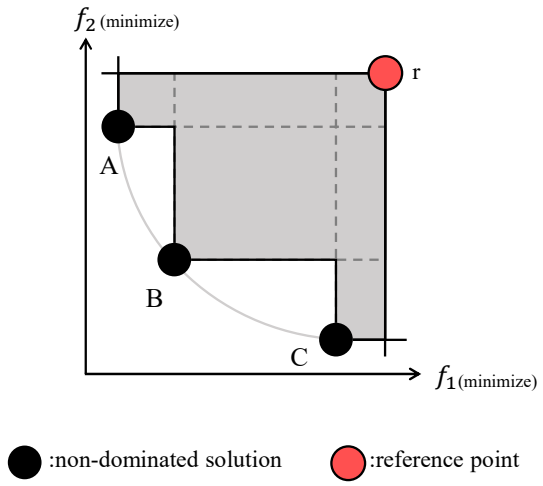

Figure 8. Calculation of hypervolume.

3.2.2. Diversity Metric

The diversity metric evaluates the equality of the distribution within the solution set. Initially, boundary solutions are recognized in multi-objective functions. Then, the Euclidean distance between each solution and its neighboring solutions is calculated, yielding a set of distance values (d_i). Average ($\overline{d}$) and difference ($d_i - \overline{d}$) values are derived from this set. Lastly, Equation (19) is utilized to compute the diversity index of the solution set.

$$\text{diversity metric} = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \overline{d} \right|}{d_f + d_l + (N-1)\overline{d}}, \tag{19}$$

3.2.3. The Number of Retained Non-Dominated Solutions

This study utilizes non-dominated sorting [14] to assess the Pareto solution sets generated by different methods. The process is delineated in Figure 9. All Pareto solution sets from each technique are initially collected, and non-dominated solutions are preserved. Subsequently, the non-dominated solutions across all methods are amalgamated, uniquely identifying each method's solutions. Then, all solutions undergo non-dominated sorting. After sorting, each solution receives a rank, and those not at rank 1 are removed, leaving only dominant solutions. Finally, the quantity of retained non-dominated solutions per method is determined based on their assigned numbers.

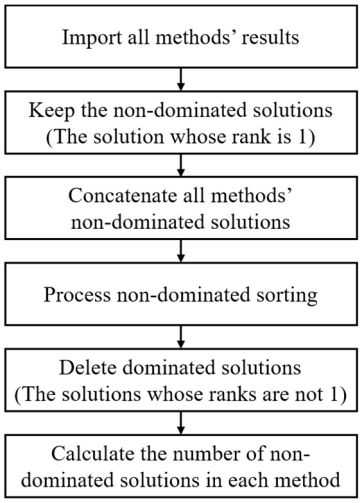

Figure 9. Non-dominated solution process.

3.3. Experimental Parameters

Tables 2 and 3 below illustrate the pertinent parameters for edge computing, the multi-objective bat algorithm (MOBA), and reinforcement learning (SARSA).

Table 2. Configuring parameters for the MOBA.

Parameters	Value
Total number of SeNBs in the system, M	5
Total number of SMDs in the system, S	98
Transmission power of SMD ($U_{m,j}$), $p_{m,j}$	{4, 5, 6} W
Power consumption coefficient of SMD ($U_{m,j}$), $\varepsilon_{m,j}$	10^{-28}
SMD's remaining energy, $e_{m,j}$	90,000 J
Computation capability of MEC server, F	30 GHz
Computation capability of SMD ($U_{m,j}$), $f_{m,j}$	[0.5, 1] GHz
Data size of task $\tau_{m,j,k}$, $d_{m,j,k}$	[500, 1000] K bits
Number of CPU cycles required to perform a bit of the task, $CPU^{c_{m,j,k}}$	500 CPU cycles/bit
Bandwidth, B	20 MHz
Number of channels	10
The fixed altitude of base station, H	10 (m)
Noise power, σ^2	−100 (dBm)

Table 2. *Cont.*

Parameters	Value
Channel power gain at a reference distance of 1 m and a transmitting power of 1 W, g_0	0.1
Population size, npop	100
Dimension of bat, N	98
Loudness, A_i	[0, 1]
Pulse rate, r_i	[0, 1]
Initial pulse rate, r_0	0.6
f_{min}	0
f_{max}	2
Iterations	100

Table 3. Configuring parameters for the SARSA.

Parameters	Value
lr, learning rate	0.3
df, decay factor	0.9
ε of ε-greedy	[0, 1]
$\Delta\varepsilon$ of ε-greedy	0.04
ω_1 of reward	0.25
ω_2 of reward	0.75

3.4. Experimental Results

Each method undergoes 30 experiments, after which the resulting Pareto solution sets are assessed. Then, the simulations of the obtained offloading policies are performed using ns-3.

3.4.1. Edge Computing Constraints

In Table 4, MOBA-CV without reinforcement learning, while considering constraints, enhanced the hypervolume indicator by 4.68% and 1.54% compared to NSGA-II-CV and MOPSO-CV, respectively, across 30 experiments. Despite MOBA-CV displaying a higher standard deviation than the other methods, it outperformed NSGA-II-CV even in the worst experiment, achieving a hypervolume value of 10,599.12. Thus, despite more significant variability, MOBA-CV remains the most effective among the three methods. Furthermore, in Table 5, MOBA-CV showed a 2.72% improvement over NSGA-II-CV and a 1.05% improvement over MOPSO-CV regarding the diversity indicator.

Table 4. Comparative analysis of hypervolume without reinforcement learning.

Method	Hypervolume		
	Average	Percentage	Standard Deviation
MOBA-CV	10,967.27	X	190.33
NSGA-II-CV	10,453.76	4.68%	97.8
MOPSO-CV	10,798.6	1.54%	118.57

In comparing non-dominated sorting over 30 iterations, none of the methods yielded a "retained non-dominated solution count equals 0", indicating each method's successful discovery of non-dominated solutions. MOBA-CV consistently retained the highest number of

non-dominated solutions, as evidenced by the "retained non-dominated solution count equals the most" in each comparison, underscoring its superior performance, as depicted in Table 6.

Table 5. Comparative analysis of diversity metric without reinforcement learning.

Method	Diversity Metric		
	Average	Percentage	Standard Deviation
MOBA-CV	0.2098	X	0.0029
NSGA-II-CV	0.2155	2.72%	0.0025
MOPSO-CV	0.212	1.05%	0.0029

Table 6. Comparative analysis of non-dominated solutions.

Method	Times	
	Number of Non-Dominated Solutions Is Zero	Number of Non-Dominated Solutions Is Maximum
MOBA-CV	0	30
NSGA-II-CV	0	0
MOPSO-CV	0	0

Table 7 displays the outcomes of the optimal solutions identified by three methods concerning total network energy consumption and latency. Latency and energy consumption represent conflicting objectives; however, when latency is constrained within acceptable limits, MOBA-CV achieves reductions of 77.543 W and 34.722 W in energy consumption compared to NSGA-II-CV and MOPSO-CV, respectively. These results unmistakably indicate MOBA-CV's capability to attain a more harmonized solution amidst such trade-off dilemmas. Additionally, Table 8 outlines the mean computation time for each approach. MOBA-CV exhibits computations of 3.562 s and 642.416 s, making it swifter than NSGA-II-CV and MOPSO-CV, respectively. Given the context of edge computing, shorter computation time aligns better with optimizing task-offloading determinations in this domain. In Figure 10, we can clearly observe the comparison results after non-dominated sorting. MOBA-CV retains 41 non-dominated solutions, and NSGA-II-CV and MOPSO-CV have 9 and 24 solutions, respectively. Obviously, MOBA-CV not only obtains the largest number of non-dominated solutions, but more importantly, these solutions achieve a balance between delay time and energy consumption. These solutions are mainly concentrated in the center of the graph. In comparison, the solutions obtained by other comparison methods are mainly concentrated in the upper left and lower right corners. This means that although these solutions optimize one objective, they sacrifice performance for another objective.

Table 7. Latency time and energy consumption.

Method	Latency Time (s)		Energy Consumption (W)	
	Value	Difference	Value	Difference
MOBA-CV	147.1	2.81	594	X
NSGA-II-CV	144.3	X	671.5	77.54
MOPSO-CV	145.5	1.2	628.7	34.72

Table 8. Comparative analysis of average execution time without reinforcement learning.

Method	Average Execution Time (s)
MOBA-CV	7.373
NSGA-II-CV	10.935
MOPSO-CV	649.789

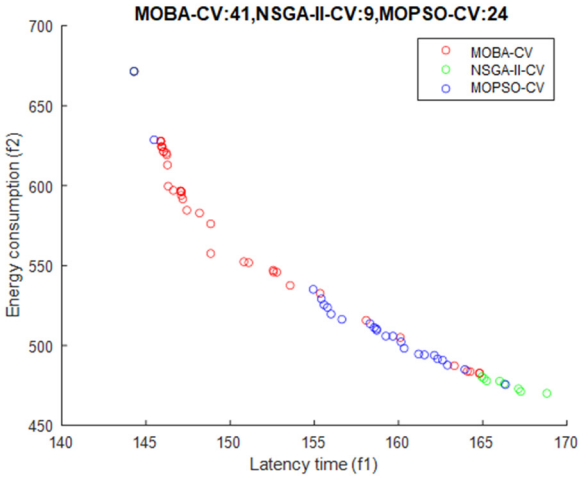

Figure 10. Comparison of the number of retained non-dominated solutions in edge computing constraint experiments.

3.4.2. Constraints in Edge Computing and the Application of Reinforcement Learning

In Table 9, MOBA-CV-SARSA shows a 0.9% improvement in the hypervolume indicator compared to NSGA-RL-CV. In contrast, its improvement over QLPSO-CV reaches 15.07%. In 30 comparative experiments, MOBA-CV-SARSA exhibits a slightly higher standard deviation than NSGA-RL-CV. Although MOBA-CV-SARSA's worst performance yields a hypervolume value of 10,771.17, somewhat lower than NSGA-RL-CV's average, it still outperforms QLPSO-CV. In Table 10, MOBA-CV-SARSA's diversity indicator surpasses NSGA-RL-CV by 4.72% and QLPSO-CV by 0.1%. In Table 11, both MOBA-CV-SARSA and NSGA-RL-CV never have zero retained non-dominated solutions in 30 comparisons. QLPSO-CV has this issue seven times, indicating SARSA's superiority over Q-learning. Under the "most retained non-dominated solutions" scenario, MOBA-CV-SARSA consistently preserves the most solutions, thus achieving the best overall performance. Table 12 illustrates the average computation time for each method. Notably, MOBA-CV-SARSA computes 64.338 s faster than NSGA-RL-CV and 200.524 s faster than QLPSO-CV.

Table 9. Comparative analysis of hypervolume with reinforcement learning.

Method	Hypervolume		
	Average	Percentage	Standard Deviation
MOBA-CV-SARSA	11,008.45	X	143.85
NSGA-RL-CV	10,909.92	0.9%	51.51
QLPSO-CV	9348.99	15.07%	138.77

Table 10. Comparative analysis of diversity metric.

Method	Diversity Metric		
	Average	Percentage	Standard Deviation
MOBA-CV-SARSA	0.2098	X	0.0092
NSGA-RL-CV	0.2197	4.72%	0.0016
QLPSO-CV	0.21	0.1%	0.0027

Table 11. Comparative analysis of 0 and MAX with reinforcement learning.

Method	Diversity Metric		
	Average	**Percentage**	**Standard Deviation**
MOBA-CV-SARSA	0.2098	X	0.0092
NSGA-RL-CV	0.2197	4.72%	0.0016
QLPSO-CV	0.21	0.1%	0.0027

Table 12. Comparative analysis of average execution time with reinforcement learning.

Method	Average Execution Time (s)
MOBA-CV-SARSA	61.018
NSGA-RL-CV	125.356
QLPSO-CV	261.542

3.4.3. Dynamic Range Adjustment

In this section, we explore the impact of dynamic tuning and its outperformance. Initially, MOBA-CV is introduced, focusing solely on constraint conditions. The optimal α and γ values for this approach are determined through 8250 iterations of hyperparameter experiments, complemented by the optimal range values as referenced by Yang et al. [15]. Subsequently, we introduce MOBA-CV-SARSA with a fixed range, utilizing a predefined search range, and MOBA-CV-SARSA-MA, which employs manual adjustment for a narrower search scope. Lastly, MOBA-CV-SARSA-DA dynamically adjusts the parameter ranges, incorporating an automatic switching mechanism. Table 13 presents various methods and their respective parameter configurations for the comparative experiment.

Table 13. Configuring parameters for the MOBA with dynamic range adjustment.

Method	Parameters
MOBA-CV	α: 0.03, γ: 0.93 (8250 runs of hyperparameter experiment)
	α: 0.9, γ: 0.9 (reference [15])
MOBA-CV-SARSA with fixed range	α: [0, 1], γ: [0, 1] α, γ difference is 0.01
MOBA-CV-SARSA-MA	α: $[\lfloor \alpha^* \rfloor, \lfloor \alpha^* \rfloor + 0.1]$, γ^*: $[\lfloor \gamma^* \rfloor, \lfloor \gamma^* \rfloor + 0.1]$ α, γ difference is 0.006
MOBA-CV-SARSA-DA	broad search
	α: [0, 1], γ: [0, 1] α, γ difference is 0.01
	narrow search
	α: $[\lfloor \alpha^* \rfloor, \lfloor \alpha^* \rfloor + 0.1]$ γ: $[\lfloor \gamma^* \rfloor, \lfloor \gamma^* \rfloor + 0.1]$ α, γ difference is 0.006

The hypervolume indicator in Table 14 illustrates that under the sole consideration of edge computing constraints, MOBA-CV achieves superior performance when optimized with α and γ values derived from experimental design, surpassing the performance of the MOBA-CV referenced in [15]. Furthermore, as in MOBA-CV-SARSA, augmenting MOBA-CV with reinforcement learning enhances its performance compared to MOBA-CV without such augmentation. Notably, MOBA-CV-SARSA-MA, which utilizes optimal solutions α^* and γ^* obtained from 100 searches of MOBA-CV-SARSA with a fixed range, exhibits a 0.11% improvement over MOBA-CV-SARSA with a fixed range. Since MOBA-CV-SARSA-MA leverages the search outcomes of MOBA-CV-SARSA with a fixed range,

this study integrates both approaches. It introduces an automatic switching mechanism in MOBA-CV-SARSA-DA.

Table 14. Comparative analysis of hypervolume with dynamic range adjustment.

Method		Hypervolume		
		Average	**Percentage**	**Standard Deviation**
The total number of iterations is 100 times.				
MOBA-CV	the best α, γ	10,967.27	0.58%	190.33
	reference [15]	10,950.75	0.73%	195.63
MOBA-CV-SARSA with fixed range		11,008.45	0.20%	143.85
The total number of iterations is 200 times.				
MOBA-CV-SARSA-MA		11,020.47	0.1%	134.63
MOBA-CV-SARSA-DA		11,031.05	X	214.02
The total number of iterations is less than 200 times.				
MOBA-CV-SARSA-DA		11,023.87	0.07%	216.49

MOBA-CV-SARSA-DA employs two distinct strategies: (1) a total of 200 iterations for both fixed-range and small-range searches, and (2) 100 iterations for fixed-range searches combined with the same for small-range searches, resulting in a total of fewer than 200 iterations. Our analysis of the results reveals that when the total iteration count reaches 200, MOBA-CV-SARSA-DA exhibits a 0.095% enhancement in the hypervolume indicator compared to MOBA-CV-SARSA-MA. Moreover, MOBA-CV-SARSA-DA achieves performance akin to MOBA-CV-SARSA-MA without necessitating the completion of 200 iterations. In Table 15, MOBA-CV-SARSA with a fixed range showcases the most favorable diversity metric. Its broader search range provides increased flexibility in adjusting solutions, facilitating the exploration of a more comprehensive array of solutions.

Table 15. Comparative analysis of diversity metric.

Method		Diversity Metric		
		Average	**Percentage**	**Standard Deviation**
The total number of iterations is 100 times.				
MOBA-CV	the best α, γ	0.2098	0.43%	0.0029
	reference [15]	0.2101	0.57%	0.0023
MOBA-CV-SARSA with fixed range		0.2089	X	0.0092
The total number of iterations is 200 times.				
MOBA-CV-SARSA-MA		0.2101	0.72%	0.0030
MOBA-CV-SARSA-DA		0.2095	0.29%	0.0101
The total number of iterations is less than 200 times.				
MOBA-CV-SARSA-DA		0.2094	0.24%	0.0101

In Figures 11 and 12, we analyze the convergence curves of MOBA-CV-SARSA-DA and MOBA-CV-SARSA-MA, comparing their performance using various indicators. Regarding hypervolume, MOBA-CV-SARSA-DA not only achieves a faster convergence rate than MOBA-CV-SARSA-MA, thanks to its switching mechanism, but also displays superior overall performance. Both methods demonstrate substantial solution diversity in the early iterations when considering the diversity metric. Table 16 illustrates the results of MOBA-CV derived from α, γ hyperparameter experiments, and MOBA-CV-SARSA with fixed range using reinforcement learning, both surpassing the MOBA-CV proposed by Yang et al. [15] based on optimal intervals of α and γ. In Table 17, both methods successfully identify non-dominated solutions across 30 experiments. Notably, MOBA-CV-SARSA-DA consistently maintains a higher count of non-dominated solutions.

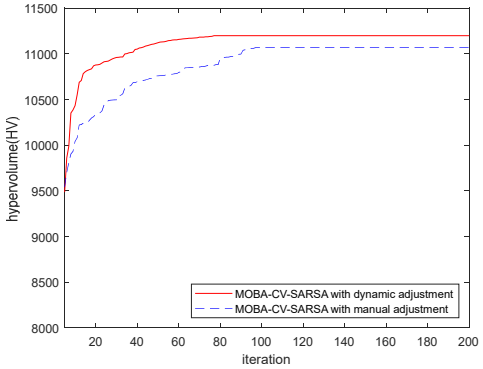

Figure 11. The convergence curve of the hypervolume.

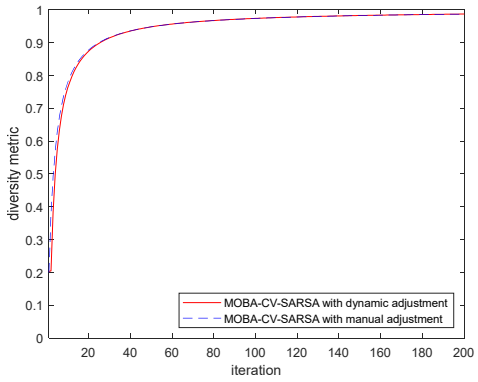

Figure 12. The convergence curve of the diversity metric.

Table 16. Comparative analysis of 0 and MAX with dynamic range adjustment.

Method		Times	
		Number of Non-Dominated Solutions Is Zero	**Number of Non-Dominated Solutions Is Maximum**
The total number of iterations is 100 times.			
MOBA-CV	the best α, γ	0	12
	reference [15]	1	6
MOBA-CV-SARSA with fixed range		0	12

Table 17. Comparative analysis of 0 and MAX.

Method	Times	
	Number of Non-Dominated Solutions Is Zero	**Number of Non-Dominated Solutions Is Maximum**
The total number of iterations is 200 times.		
MOBA-CV-SARSA-MA	0	13
MOBA-CV-SARSA-DA	0	17

In Table 18, despite MOBA-CV-ref exhibiting shorter latency than MOBA-CV-SARSA with a fixed range based on the reference settings, it consumes 195.314 W and 119.048 W more energy than the MOBA-CV derived from hyperparameter experiments. This vividly illustrates the performance enhancement achieved through reinforcement learning.

Table 18. Comparative analysis of latency and consumption with/without reinforcement learning.

Method	Latency Time (s)		Energy Consumption (W)	
	Value	Difference	Value	Difference
MOBA-CV with the best α, γ	150.8	6.525	552.4	76.236
MOBA-CV with reference [15]	144.3	X	671.5	195.314
MOBA-CV-SARSA with fixed range	151	6.7	476.2	X

Table 19 showcases that when the delay remains within an acceptable range, MOBA-CV-SARSA-DA reduces energy consumption by 102.542 W compared to MOBA-CV-SARSA-MA. This underscores the advantages of employing an automatic switching mechanism for dynamically adjusting the search scope, validating its capacity to discover more balanced solutions between the two objectives. Table 20 presents the average computational time for each method. MOBA-CV-SARSA-MA requires extended computation time based on the outcomes of MOBA-CV-SARSA with a fixed range. Conversely, MOBA-CV-SARSA-DA, integrating an automatic switching mechanism for dynamic range adjustment, saves 48.613 s compared to MOBA-CV-SARSA-MA while maintaining comparable performance. This underscores the efficacy and efficiency of incorporating an automatic switching mechanism for dynamic range adjustment in optimization problems.

Table 19. Comparative analysis of latency and consumption with/without dynamic adjustment.

Method	Latency Time (s)		Energy Consumption (W)	
	Value	Difference	Value	Difference
MOBA-CV-SARSA-MA	144.3	X	671.5	102.542
MOBA-CV-SARSA-DA	148.4	4.066	569	X

Table 20. Comparative analysis of average execution time for different tuning mechanisms.

Method		Average Execution Time (s)
The total number of iterations is 100 times.		
MOBA-CV	the best α, γ	7.373
	reference [15]	7.317
MOBA-CV-SARSA with fixed range		61.018
The total number of iterations is 200 times.		
MOBA-CV-SARSA-MA		181.881
MOBA-CV-SARSA-DA		180.434
The total number of iterations is less than 200 times.		
MOBA-CV-SARSA-DA		131.821

4. Conclusions

This study introduces an advanced learning approach, the multi-objective bat algorithm with reinforcement learning (MOBA-CV-SARSA), designed to optimize task-offloading decisions in edge computing. This optimization achieves a harmonious balance between network latency and energy consumption. By adhering to edge computing constraints and employing a tailored multi-objective function, task-offloading decisions for all devices can be concurrently determined. Compared to NSGA-RL-CV and QLPSO-CV, which integrate enhanced learning and share similar constraints, MOBA-CV-SARSA exhibits enhancements of 0.9% and 15.07% in hypervolume and 4.72% and 0.1% in the diversity metric. Additionally, it notably diminishes network energy usage within acceptable

latency limits. Leveraging enhanced learning for fine-tuning hyperparameters aids the MOBA in discovering optimal solutions. MOBA-CV-SARSA-DA outperforms its counterparts at the exact iteration count, validating the efficacy of dynamic adjustment ranges. Ultimately, the enhanced-learning-driven MOBA streamlines complex hyperparameter experiments and identifies the best trade-off solutions under set constraints.

Author Contributions: Conceptualization, C.-L.T. and C.-S.C.; methodology, C.-S.C.; software, Y.-H.S.; validation, C.-L.T., C.-S.C. and Y.-H.S.; formal analysis, C.-L.T.; investigation, C.-S.C.; resources, Y.-H.S.; data curation, C.-S.C.; writing—original draft preparation, Y.-H.S.; writing—review and editing, C.-S.C.; visualization, Y.-H.S.; supervision, C.-L.T.; project administration, C.-S.C.; funding acquisition, C.-S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Project name: Integrated Development of Multimedia Advertising Marketing Combined with AI Data Analysis grant number 210A024.

Institutional Review Board Statement: Not applicable for studies not involving humans or animals.

Informed Consent Statement: Not applicable.

Data Availability Statement: Dataset available on request from the authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, Y.; Zhang, N.; Zhang, Y.; Chen, X. Dynamic computation offloading in edge computing for Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4242–4251. [CrossRef]
2. Cui, L.; Xu, C.; Yang, S.; Huang, J.Z.; Li, J.; Wang, X.; Ming, Z.; Lu, N. Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4791–4803. [CrossRef]
3. Bozorgchenani, A.; Mashhadi, F.; Tarchi, D.; Monroy, S.A.S. Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. *IEEE Trans. Mob. Comput.* **2021**, *20*, 2992–3005. [CrossRef]
4. Gedawy, H.; Habak, K.; Harras, K.A.; Hamdi, M. RAMOS: A resource-aware multi-objective system for edge computing. *IEEE Trans. Mob. Comput.* **2021**, *20*, 2654–2670. [CrossRef]
5. Alfakih, T.; Hassan, M.M.; Al-Razgan, M. Multi-objective accelerated particle swarm optimization with dynamic programing technique for resource allocation in mobile edge computing. *IEEE Access* **2021**, *9*, 167503–167520. [CrossRef]
6. Xiao, Z.; Shu, J.; Jiang, H.; Lui, J.C.; Min, G.; Liu, J.; Dustdar, S. Multi-objective parallel task offloading and content caching in D2D-aided MEC Networks. *IEEE Trans. Mob. Comput.* **2022**, *22*, 6599–6615. [CrossRef]
7. Mohan, S.; Sinha, A. Elitist non-dominated sorting directional bat algorithm (ENSdBA). *Expert Syst. Appl.* **2023**, *227*, 120292. [CrossRef]
8. Koffka, K.; Sahai, A. A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context. *Int. J. Intell. Syst. Appl.* **2012**, *4*, 23–29.
9. Kaur, A.; Kumar, K. A reinforcement learning based evolutionary multi-objective optimization algorithm for spectrum allocation in cognitive radio networks. *Phys. Commun.* **2020**, *43*, 101196–101208.
10. Liu, Y.; Lu, H.; Cheng, S.; Shi, Y. An adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC) 2019, Wellington, New Zealand, 10–13 June 2019; pp. 815–822.
11. Marco, C.; Giovanni, F.; Riccardo, G.; Raffaele, P. A comparison among reinforcement learning algorithms in financial trading systems. *SSRN Electron. J.* **2019**. [CrossRef]
12. Zeng, Q.; Du, Y.; Huang, K.; Leung, K.K. Energy-efficient resource management for federated edge learning with CPU-GPU heterogeneous computing. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7947–7962. [CrossRef]
13. Hu, S.; Li, G. Dynamic request scheduling optimization in mobile edge computing for IoT applications. *IEEE Internet Things J.* **2020**, *7*, 1426–1437. [CrossRef]
14. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multi objective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
15. Yang, X. Bat algorithm: Literature review and applications. *Int. J. Bio-Inspired Comput.* **2013**, *5*, 141–149. [CrossRef]
16. Wang, L.; Liang, Y.; Yang, J. Improved multi-objective PSO algorithm for optimization problems. In Proceedings of the 2010 IEEE International Conference on Progress in Informatics and Computing, Shanghai, China, 10–12 December 2010; pp. 195–198.
17. Mezzavilla, M.; Zhang, M.; Polese, M.; Ford, R.; Dutta, S.; Rangan, S.; Zorzi, M. End-to-End simulation of 5G mmWave networks. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2237–2263. [CrossRef]
18. Zitzler, E.; Thiele, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Trans. Evol. Comput.* **1999**, *3*, 257–271. [CrossRef]

19. Yan, J.; Li, C.; Wang, Z.; Deng, L.; Sun, D. Diversity metrics in multi-objective optimization: Review and perspective. In Proceedings of the 2007 IEEE International Conference on Integration Technology, Shenzhen, China, 20–24 March 2007; pp. 553–557.
20. Cohen, J. *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed.; Routledge: New York, NY, USA, 1998.