sensors

MDPI

Article

MAARS: Multiagent Actor–Critic Approach for Resource Allocation and Network Slicing in Multiaccess Edge Computing

Ducsun Lim and Inwhee Joe *

Department of Computer Software, Hanyang University, Seoul 04763, Republic of Korea; imcoms@hanyang.ac.kr
* Correspondence: iwjoe@hanyang.ac.kr

Abstract: This paper presents a novel algorithm to address resource allocation and network-slicing challenges in multiaccess edge computing (MEC) networks. Network slicing divides a physical network into virtual slices, each tailored to efficiently allocate resources and meet diverse service requirements. To maximize the completion rate of user-computing tasks within these slices, the problem is decomposed into two subproblems: efficient core-to-edge slicing (ECS) and autonomous resource slicing (ARS). ECS facilitates collaborative resource distribution through cooperation among edge servers, while ARS dynamically manages resources based on real-time network conditions. The proposed solution, a multiagent actor–critic resource scheduling (MAARS) algorithm, employs a reinforcement learning framework. Specifically, MAARS utilizes a multiagent deep deterministic policy gradient (MADDPG) for efficient resource distribution in ECS and a soft actor–critic (SAC) technique for robust real-time resource management in ARS. Simulation results demonstrate that MAARS outperforms benchmark algorithms, including heuristic-based, DQN-based, and A2C-based methods, in terms of task completion rates, resource utilization, and convergence speed. Thus, this study offers a scalable and efficient framework for resource optimization and network slicing in MEC networks, providing practical benefits for real-world deployments and setting a new performance benchmark in dynamic environments.

Keywords: multiaccess edge computing; reinforcement learning; network slicing; resource allocation

check for updates

Citation: Lim, D.; Joe, I. MAARS: Multiagent Actor–Critic Approach for Resource Allocation and Network Slicing in Multiaccess Edge Computing. *Sensors* **2024**, *24*, 7760. https://doi.org/10.3390/s24237760

Academic Editor: Hai Dong

Received: 23 September 2024
Revised: 28 November 2024
Accepted: 2 December 2024
Published: 4 December 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Advancements in geospatial remote sensing technologies, driven by increased spectral, spatial, and temporal resolutions [1,2], have revolutionized various fields, including national economies, ecological protection, and national security. These technologies enable precise observations and analyses for diverse applications, including Earth observation [3] and the Internet of Things (IoT) [4], significantly improving the accuracy of information collection and analysis. Real-time processing of high-resolution data is now crucial for applications such as climate change monitoring, natural disaster response, and infrastructure management. This need for real-time analysis aligns perfectly with the capabilities of multiaccess edge computing (MEC), a transformative approach to handling time-sensitive data.

Recent breakthroughs in artificial intelligence, edge computing, and the IoT have enabled more sophisticated analyses of geographic phenomena, earth science processes, and intelligent decision-making. In particular, MEC [5] has emerged as a key technology, enhancing real-time data processing and response times by bringing computation and storage closer to the network edge. This shift has facilitated faster analysis of remote sensing data, crucial for the sustainable development of nations and cities [6–8]. However, efficient resource allocation in MEC networks, especially with high traffic and dynamic user demands, remains a significant challenge.

MEC networks provide highly efficient, real-time computing services for smart device users (SDUs). By processing tasks locally at edge nodes (ENs) instead of relying on

centralized clouds, MEC minimizes latency and conserves bandwidth [9], offering significant advantages in environments with large-scale IoT deployments, such as smart cities, autonomous vehicles, and industrial automation systems.

However, a major challenge in MEC architectures is the efficient allocation and management of the limited network resources. Network-slicing (NS) techniques address this challenge by logically partitioning the network and allocating resources based on service requirements, ensuring ultralow latency and high reliability. NS effectively virtualizes physical infrastructure, enabling flexible adaptation to diverse user needs [10].

Beyond computational resources, spectrum and power allocation are critical aspects of resource management in MEC networks. Effective power allocation optimizes energy efficiency and mitigates interference, a particularly complex task in high-traffic environments. Additionally, spectrum allocation ensures efficient use of available bandwidth to support diverse user requirements. Recent research, such as that by Ngene et al. [11], highlighted the importance of these factors, especially power allocation strategies, in resource management frameworks for next-generation communication systems.

Reinforcement learning (RL)-based resource allocation and NS algorithms have attracted considerable attention for their ability to dynamically adjust resource allocation in response to network-state changes, effectively handling unpredictable traffic patterns and user demands [12]. Various RL algorithms, such as Q-learning [13], deep Q-network (DQN) [14], policy gradient [15], and proximal policy optimization (PPO) [16], have been actively studied to optimize network-resource management.

This study proposes a novel approach to address NS and resource allocation challenges in MEC networks, aiming to maximize the success rate of user-computing tasks in each network slice through real-time monitoring and efficient network-resource allocation. Specifically, these problems are decomposed into two subproblems, i.e., efficient core-to-edge slicing (ECS) and autonomous resource slicing (ARS), and a multiagent actor–critic resource scheduling (MAARS) algorithm is proposed to handle them. MAARS combines MADDPG for ECS and SAC for ARS to optimize resource distribution and real-time resource management, respectively.

This integration of MADDPG and SAC creates a unique two-tier optimization framework tailored for MEC networks. MADDPG effectively manages multiagent coordination in distributed environments, enabling precise core-to-edge resource allocation for ECS. SAC, employed for ARS, provides stable and efficient real-time decision-making in scenarios with continuous action spaces. The synergistic combination of these algorithms ensures seamless resource management across the entire network.

To further enhance the performance of MADDPG, we introduce model-agnostic metalearning (MAML) for metalearning initialization. MAML pre-trains the MADDPG neural networks with optimized meta-policy parameters, enabling rapid adaptation to dynamic network conditions. This approach reduces training iterations and improves efficiency and scalability, ensuring robust performance under fluctuating conditions.

This innovation is critical for addressing the challenges in MEC networks, where rapid adaptation is essential. MAML equips MADDPG to handle diverse resource demands effectively, ensuring lower latency, better resource utilization, and higher task completion rates. Compared to traditional methods, this approach significantly reduces convergence time and improves the adaptability of resource allocation strategies.

Extensive experiments demonstrates the superiority of the proposed approach. The MAARS framework outperforms existing state-of-the-art methods, achieving higher user task completion success rates, improved resource utilization, and reduced convergence time. This is attributed to the integrated MADDPG and SAC framework, enabling independent yet interconnected optimization of ECS and ARS tasks.

By effectively decomposing complex resource allocation problems and applying advanced multiagent RL techniques, MAARS significantly advances MEC network management. The innovative integration of MADDPG, SAC, and MAML enhances computational

efficiency and ensures robust and dynamic adaptation to evolving network conditions, setting a new benchmark for MEC NS and resource allocation.

The remainder of this paper is organized as follows. Section 2 reviews the related research in this field, and Section 3 presents the system model and mathematical formulation of the problem. Section 4 describes the proposed algorithm and experimental conditions, Section 5 presents the simulation results for evaluating the algorithm performance, and Section 6 discusses them. Finally, Section 7 concludes the study and presents directions for future research.

2. Related Work

NS and resource allocation in MEC environments have been extensively studied [17–28]. Researchers have explored various approaches to optimize these processes. For instance, Tran et al. [17] proposed a mixed-integer nonlinear programming approach with heuristic algorithms to optimize task offloading and resource allocation in MEC networks, improving offloading utility under resource constraints. However, it faces challenges in adapting to dynamic network conditions in real-time. To overcome this limitation, Cevallos et al. [18] introduced a dueling double deep Q-network (D3QN) for 5G content delivery networks, enhancing adaptability to network complexity and reducing operational costs.

Deep reinforcement learning (DRL) has also emerged as a promising solution. Huang et al. [19] developed a DRL-based offloading algorithm to minimize energy consumption in mobile devices while meeting real-time task deadlines; however, scalability remained a concern when handling numerous simultaneous tasks. Similarly, Wang et al. [20] used DRL for dynamic resource allocation and service time reduction, but encountered challenges associated with learning speed and stability. Nduwayezu et al. [21] addressed subcarrier allocation in non-orthogonal multiple access (NOMA) MEC systems using a DRL-based task offloading algorithm, achieving significant improvements in resource utilization and computation speed. Ning et al. [22] further extended this work by focusing on resource allocation in multiuser environments, enabling dynamic MEC server–user interactions to reduce computation costs.

Slicing frameworks have proven crucial for enhancing network scalability and performance. Shah et al. [23] and Khan et al. [24] developed frameworks for scalable dedicated networks and dynamic resource adjustments based on traffic criticality. Wu et al. [25] proposed a two-tier constraint algorithm for resource allocation in vehicular-communication systems to ensure reliable services in high-traffic conditions. Seah et al. [26] introduced a latency-aware resource scheduler to minimize latency while maintaining system fairness. Jin et al. [27] and Chiang et al. [28] applied deep RL techniques to resource slicing, reducing latency and enhancing computational efficiency in MEC systems.

Building upon these advancements, this paper introduces the MAARS algorithm, a novel two-stage slicing model designed to address NS and resource-allocation challenges in MEC environments. MAARS employs multiagent RL techniques—MADDPG for ECS and SAC for ARS—to enable dynamic and collaborative resource management. Additionally, by integrating MAML for neural network initialization, MAARS accelerates learning and enhances adaptability to network changes, thereby improving task success rates and overall system performance.

This study contributes to the field by proposing an innovative two-stage slicing model that effectively addresses NS and resource-allocation issues in MEC networks. The model combines ECS and ARS to efficiently allocate network resources while adapting to real-time demands. By leveraging RL for dynamic resource allocation, the proposed framework offers better flexibility than existing methods, maximizing network performance and significantly increasing the success rates of user-computing tasks.

3. System Model

This section describes the MEC-based network architecture, and the proposed system model based on this architecture for handling tasks.

3.1. System Architecture

Figure 1 illustrates an MEC network comprising numerous ENs and SDUs. SDUs generate different types of computing tasks to fulfill the requirements of various services, such as augmented reality, virtual reality, video streaming, and online gaming. Each EN comprises a base station (BS) and an associated EN, which provide network bandwidth and computing resources, respectively. Network function virtualization (NFV) abstracts the network infrastructure into virtual resources to enable flexible resource management, whereas software-defined networking (SDN) flexibly configures network functions to provide optimal network performance.

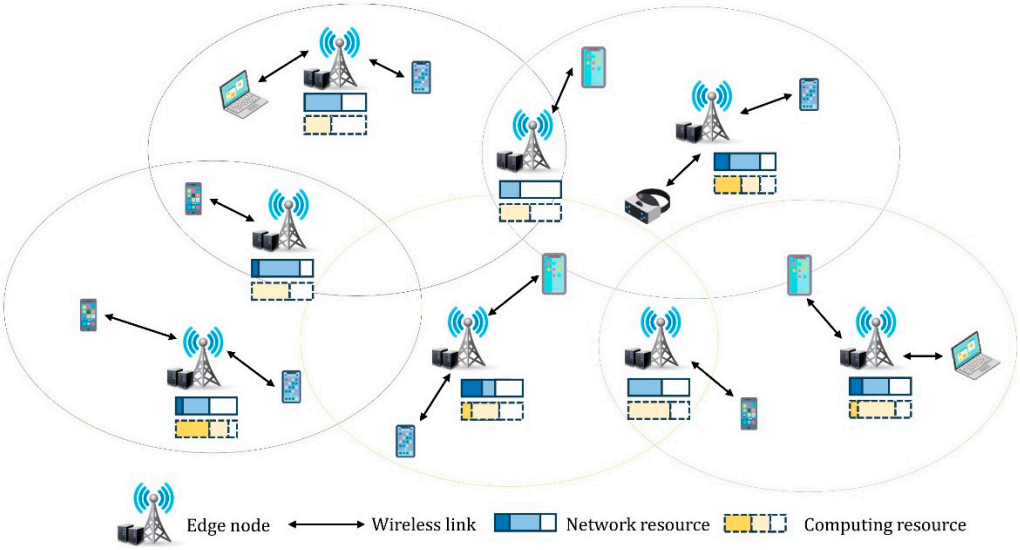

Figure 1. System architecture.

The MEC network provides logical access to all network-based SDUs. Each network has a set of BSs and a corresponding set of associated ENs. Each BS has a certain amount of network bandwidth, and each EN provides a certain amount of computing resources. The network resources comprise the bandwidth and computing resources of all BSs and ENs. Users generate W computing jobs with different service-delay requirements, which are typically offloaded to the nearest EN based on the location of the user. Each computing task can be offloaded to only one EN, and all ENs in the network exchange information to ensure optimal resource allocation.

This system model leverages NFV and SDN technologies to provide high-quality services to users through dynamic resource allocation and optimization. This increases network flexibility and scalability and maximizes MEC network performance by efficiently handling various service requirements. The notations employed in the mathematical representation of the system model are listed in Table 1.

Table 1. Mathematical notations of the system model.

Notation	Definition
$R_m(t)$	Resource-allocation set for m-th logical network at time t
$B_m(t)$	Allocated bandwidth for the m-th network at time t
$F_m(t)$	Allocated computing resources for the m-th network at time t
$C_m(t)$	Allocated caching resources for the m-th network at time t
X_m	Resource-entity set for the m-th logical network
$W_{w,\,m}(t)$	Allocated resource entity for the w-th service in the m-th network
$Svr_{k,\,m}(t)$	Server-selection indicator for task allocation
$r_k(t)$	Data-transmission rate for task k at time t

Table 1. *Cont.*

Notation	Definition
$\Gamma_{k,\,trans}(t)$	Transmission delay for task k at time t
$\Gamma_{k,\,proc}(t)$	Processing delay for task k at time t
$\Gamma_{k,\,total}(t)$	Total delay for task k at time t
$\rho_{w,m,n}(t)$	Completion ratio for service w in the m-th network slice at time t
$\rho_{m,total}(t)$	Total task-completion ratio for the m-th network slice at time t
$\rho_{avg}(t)$	Weighted average of task-processing ratio at time t
$B_{w,\,m,\,total}(t)$	Total bandwidth allocated for w-th service in the m-th network at time t
$F_{w,\,m,\,total}(t)$	Total computing resources allocated for the w-th service in the m-th network at time t
$B_n(t)$	Allocated bandwidth for the n-th network entity at time t
$F_n(t)$	Allocated computing resources for the n-th network entity at time t
$B_{network,\,max}(t)$	Maximum network bandwidth
$F_{network,\,max}(t)$	Maximum computing resources in the network
πECS	Resource-allocation policy for enhanced cognitive scheduling
πARS	Resource-allocation policy for adaptive resource scheduling
$L(\theta_i)$	Loss function for agent i
y_i	Target learning Q-value for agent i
$o_m(t)$	Local observation vector for the m-th agent at time t
$s(t)$	State vector for all agents at time t
$a_m(t)$	Action vector for the m-th agent at time t
$a(t)$	Joint-action vector for all agents at time t
$r_m(t)$	Reward function for agent m at time t
ϕ_i'	Updated policy parameters for agent i
ϕ_i^*	Optimal policy parameters for agent i
Θ	Meta-policy parameters
Θ'_{new}	Updated meta-policy parameters
$State(t)$	State vector representing the network condition at time t
$Action(t)$	Action vector representing the resource allocation at time
$reward(t)$	Reward function to optimize resource-usage at time t
ϑ_{max}	Maximum allowable delay
$\vartheta_{actual,\,i}(t)$	Actual delay for user i at time t

3.2. NS Model

At the network level, the computing and data resources are dynamically allocated according to the real-time service demand of each logical access network. To achieve this, we introduce the ECS and ARS slicing models. The ECS manages resources through cooperation between the central management system of the network and ENs and is responsible for optimization and resource distribution within the network. By contrast, the ARS allocates resources based on the real-time service demands of ENs that are physically close to users and aims to minimize latency. This model enables efficient network-resource management by the central management system and edge-computing nodes, as resources are dynamically allocated based on user demand, thereby optimizing the performance and quality of network services. The resource-allocation set for each logical access network is defined as follows:

$$R_m(t) = \{B_m(t), F_m(t),\ C_m(t)\}, \tag{1}$$

where $B_m(t)$ denotes the bandwidth allocated to the network at time t, $F_m(t)$ denotes the allocated computing resources, and $C_m(t)$ denotes the caching resources. This resource-allocation set illustrates the distribution of network bandwidth, computing resources, and caching resources over time. It represents an efficient resource-allocation process aimed at handling real-time changes in user demands. State information is dynamically collected from each EN, and a virtual controller is assumed to manage this information. Network resources are composed of individual network-resource entities, and the set of these entities is expressed as

$$X_m = \{X_m |\ m = 1,\ 2,\ \ldots, M\}, \tag{2}$$

where X is the specific resource entity assigned to the m-th logical network. Each network-resource entity is periodically partitioned based on the current demands of user-computing tasks such that resources are efficiently allocated for different user tasks. This process is expressed as follows:

$$Z_w(t) = \{B_w(t),\ F_w(t),\ C_w(t)\}, \tag{3}$$

where $B_w(t)$, $F_w(t)$, and $C_w(t)$ denote the bandwidth required for the w-th service, the computing resources required for it, and the caching resources, respectively. Equation (3) indicates that network resources are dynamically managed over time and appropriately allocated based on real-time changes in user demands. The network-resource entities are efficiently managed through a central management system and virtual controller, thereby enhancing network performance and stability. This is defined as follows:

$$X_m = \{X_{w,\,m} |\ w = 1,\ 2,\ \ldots, W\}, \tag{4}$$

where X_m represents the m-th service-resource instance generated by the m-th network-resource entity. These resource entities are dynamically adjusted according to network demands, and network resources are efficiently allocated based on the predictive models and real-time data. This enables continuous monitoring of network performance and allows real-time adjustments to resource-allocation strategies to maintain optimal network performance. For example, when the demand for a specific service increases, the necessary resources are immediately allocated to maintain service quality. Conversely, when demand decreases, resources are reclaimed and reallocated to other services, thereby maximizing the overall efficiency of network resource utilization.

3.3. Resource Allocation in Network-Resource Entities

During resource allocation, the computational tasks of each service-resource entity, Z_w, are dynamically offloaded to the most suitable nearby BS. Specifically, when a user generates a task $u_{w,\,k}(t)$, it is sent to the nearest BS that can handle it efficiently. At this point, the network continuously monitors the status of computing resources in real-time to ensure optimal resource allocation. This dynamic offloading mechanism enables efficient resource management, thereby ensuring both task completion and network stability.

The BS allocates network-resource units (NRUs) to handle computing tasks and transmits the task data to servers connected to the BS. An NRU comprises the bandwidth, network links, and other network-related resources optimally distributed by the resource-management system. The server then allocates a computing resource unit (CRU) to handle the task. A CRU comprises computing resources, such as a central processing unit (CPU), graphics processing unit (GPU), or memory, and the resource-management system of the server allocates the appropriate resources to meet the task requirements.

The server utilizes its allocated CRUs to process computing tasks and sends the results back to the user. During this process, resource usage is constantly monitored, and resources are redistributed as required to maximize system efficiency. This resource-management process ensures that network and computing resources are maintained under optimal conditions, and users experience low latency and are provided with highly reliable services.

3.4. Delay Analysis in Network-Resource Management

The computational tasks of the user are dynamically offloaded to the network-slice segment (NSS) and reconfigured to optimize overall system performance. The k-th computing task of a user is represented as a tuple, $u_{w,\,k}(t) = \left(V_k(t),\ P_k(t),\ resp_{w,max}\right)$, where $v_k(t)$, $P_k(t)$, and $resp_{w,max}$ denote the data volume, computational complexity, and maximum allowable response time, respectively. Note that defining the performance requirements of each task is essential for precise resource allocation.

The server-selection indicator, $Svr_{k,m}(t) \in \{0, 1\}$, determines the MEC server to which a specific task $\phi_{wk}(t)$ is offloaded. $S_{km}(t) = 1$ indicates that the task is offloaded to a particular server, whereas $S_{km}(t) = 0$ indicates that it is not. To optimize performance, the

computational tasks of each user are offloaded to a single MEC server, thus satisfying the following constraint:

$$\sum_{m=1}^{M} Svr_{k,m}(t) \leq 1. \tag{5}$$

This ensures each task is assigned to only one MEC server, thereby preventing redundant resource allocation and promoting efficient resource usage. The number of network resources allocated to task $\phi_{wk}(t)$ is denoted as $B_{k.w,m,n}(t)$, and the transmission rate for the task is defined according to Shannon's formula as follows:

$$\mathscr{R}_k(t) = \sum_{n=1}^{N} \left[S_{km(t)} \cdot B_{k.w,m,n}(t) \cdot \log_2 \left(1 + \frac{p_k(t) h_{k,m}(t)}{N_0 + I_{km}} \right) \right], \tag{6}$$

where $p_k(t)$ represents the transmission power, $h_{k,m}(t)$ denotes the channel gain, N_0 is the noise power, and I_{km} denotes interference based on the network-resource state. This equation plays a key role in optimizing the transmission rate for efficient network-resource management. The transmission delay $\Gamma_{k,\,trans}(t)$ is defined as a function of the data size to be transmitted and the transmission rate, as follows:

$$\Gamma_{k,\,trans}(t) = \frac{V_k(t)}{r_k(t)}, \tag{7}$$

where $V_k(t)$ represents the data size and $r_k(t)$ denotes the transmission rate. Equation (7) indicates that a lower transmission delay implies higher performance. Next, the processing delay $\Gamma_{k,proc}(t)$ can be defined based on the task complexity and allocated computing resources as follows:

$$\Gamma_{k,proc}(t) = \frac{W_k(t)}{F_{k,w,m,n}(t) \cdot \eta(t)} + \varsigma(t) \cdot \frac{\rho_{k,\,n}(t)}{M_k(t)}, \tag{8}$$

where $W_k(t)$ represents the amount of work to be processed and $F_{k,l,m,n}(t)$ denotes the allocated computing resources. Additionally, $\eta(t)$ is a coefficient representing the processing efficiency of the node, whereas $\varsigma(t)$ denotes for the additional computational overhead, reflecting the extra time incurred during computation. Furthermore, $\rho_{k,\,n}(t)$ represents the task wait time based on the network conditions. Finally, $M_k(t)$ indicates the number of cores used for parallel processing or resource availability, thereby ensuring that processing delays are managed according to the resource availability. The total service delay Γ_k comprises several components and is defined as follows:

$$\Gamma_{k,\,total}(t) = \Gamma_{k,prep,\,trans}(t) + \Gamma_{k,\,trans}(t) + \Gamma_{k,\,queue,\,trans}(t) + \Gamma_{k,\,prep,proc}(t) + \Gamma_{k,\,queue,proc}(t) + \Gamma_{k,proc}(t). \tag{9}$$

These include the transmission-preparation, transmission, waiting, computation-preparation, computation-waiting, and processing delays. If the total service delay is less than the maximum allowable response time, the task is considered successfully processed. This allows for sophisticated performance evaluation that considers all possible delay factors based on the network state and available computing resources. The ratio of successfully processed tasks to the total number of processed tasks is defined as the success rate and formulated as follows(Appendix A):

$$\rho_{w,m,n}(t) = \frac{\mathscr{U}_{success,w,m,n}(t)}{\mathscr{U}_{total,w,m,n}(t)} \cdot \left(\frac{1}{1 + \exp(-o \cdot (\mathscr{R}_{w,m,n}(t) - \mathscr{R}_{th}} \right), \tag{10}$$

where $\mathscr{U}_{success,w,m,n}(t)$ denotes the numbers of successfully processed tasks and $\mathscr{U}_{total,l,m,n}(t)$ those processed during the given time period. $R_{w,m,n}(t)$ reflects the state of the resources

allocated to the w-th task, R_{th} is the resource threshold, and o adjusts the impact of resource status on the success rate. Finally, the success rate of each NSS is defined as follows:

$$\rho_{m,total}(t) = \frac{\sum_{w=1}^{W}\sum_{j=1}^{N_{w,m}} \alpha_{w,m,j} \cdot \mathcal{U}_{success,w,m,j}(t)}{\sum_{w=1}^{W}\sum_{j=1}^{N_{w,m}} \mathcal{U}_{total,w,m,j}(t)}, \tag{11}$$

where $\alpha_{w,m,j}$ is a weighting coefficient that reflects the importance of each resource and can be adjusted based on the resource status or network load. By using this weighted system, more emphasis can be placed on critical resources to enhance the resource-allocation and task-processing efficiency. The weighted average of the success rate for the tasks in each NS segment is defined as follows:

$$\rho_{avg}(t) = \sum_{m=1}^{M} \rho_{m,total}(t) \cdot \beta_m(t), \tag{12}$$

where $\beta_m(t)$ is the weighting coefficient for each NSS and is set to reflect the importance and priority of each segment. It provides critical information for comparing performance across network slices and aids in establishing appropriate resource allocation and optimization strategies.

3.5. Problem Formulation

This study aims to address NS and resource-allocation issues in MEC networks. The objective was to maximize the weighted average completion rate of users' computational tasks based on the total number of network and computing resources. Thus, the tasks of each network-slice entity are processed through NS and resource allocation while meeting the service requirements of each user. To achieve this, it is assumed that the network-slice entities are provided, and the focus is on the periodic resource-allocation problem for different types of computational tasks. This problem can be formulated as follows:

$$P: \max_{B,F} \lim_{T\to\infty} \frac{1}{T} = \sum_{t=1}^{T} \rho_{avg}(t), \tag{13}$$

and is subject to the following constraints:

$$\rho_k(t) \le \rho_{max},\ \forall k \in \{1, \ldots, \mathcal{U}_{w,m.n}\},\ \forall w \in \{1, \ldots, W\},\ \forall m \in \{1, \ldots, M\},\ \forall n \in \{1, \ldots, N\}, \tag{13a}$$

$$B_{w,m}(t) \le B_{w,m,total}(t),\ F_{w,m}(t) \le F_{w,m,total}(t),\ \forall w,\ m, \tag{13b}$$

$$B_n(t) \le B_{n,max},\ F_n(t) \le F_{n,max},\ \forall n, \tag{13c}$$

$$\sum_{m=1}^{M} B_{w,m}(t) = B_{w,\,total}(t),\ \sum_{m=1}^{M} F_{w,m}(t) = F_{w,\,total}(t) \forall w \in \{1, \ldots, W\},\ \forall m \in \{1, \ldots, M\}, \tag{13d}$$

$$\sum_{n=1}^{N} B_n(t) \le B_{network,\,max},\ \sum_{n=1}^{N} F_n(t) \le F_{network,\,max},\ \forall n \in \{1, \ldots, N\}. \tag{13e}$$

These constraints include critical factors for ensuring the efficient allocation and management of network resources:

- Constraint (13a) ensures that the service delay for each user task satisfies the delay requirements of the network; thus, it is key to maintaining user experience and network performance.
- Constraint (13b) ensures that each network slice operates efficiently with the resources allocated to it, thereby preventing resource wastage and enhancing system efficiency.
- Constraint (13c) guarantees that resources are not overused by each network node, thereby maintaining network stability and reliability.

- Constraint (13d) balances resource allocation across network-slice entities, prevents the concentration of resources at specific slices, and ensures fair resource distribution.
- Finally, Constraint (13e) ensures that total network-resource usage does not exceed available resources and manages and optimizes the overall network performance.

These constraints effectively address the NS and resource-allocation issues in an MEC network and play a key role in maximizing its performance and stability.

4. RL-Based NS and Resource-Allocation Algorithm

4.1. Overview

Because ECS and ARS operate on different time scales, solving the entire problem using existing algorithms is challenging. Therefore, it is decomposed into two subproblems: ECS (SP1) and ARS (SP2). SP1 periodically conducts ECS for each network-slice entity to maximize the weighted average completion rate of users' computational tasks. This ensures the efficient use of network resources and optimizes the user experience and can be expressed as follows:

$$SP1: \max_{\pi ECS} \mathbb{E}\left[\sum_{t=1}^{T} \alpha_{new} \cdot v\mathrm{ECS}(t)\right], \tag{14}$$

where πECS represents the resource-allocation policy for conducting ECS within a network-slice instance and $vECS(t)$ denotes the reward calculated at time t based on the resource-allocation efficiency and success rate of user computational tasks. This equation reflects the essence of ECS and provides a direction for maximizing NS effectiveness.

By contrast, the objective of SP2 is to allocate resources for each computational task such that network-resource utilization is maximized at the ENs associated with the service-level slice instances. This problem can also be addressed using RL and is formulated as follows:

$$SP2: \max_{\pi ARS} \mathbb{E}\left[\sum_{t=1}^{T} v\mathrm{ARS}(t)\right], \tag{15}$$

where πARS represents the resource-allocation policy for conducting ARS at the ENs and $vARS(t)$ denotes the reward calculated at time t based on the network-resource utilization. This equation aims to optimize resource utilization at the ENs, enhance overall network performance, and minimize resource wastage.

Therefore, we propose two algorithms to solve the network slicing and computing resource-allocation problems: MADDPG-based ECS and SAC-based ARS. The ECS algorithm optimizes resource scheduling for each network-slice entity, whereas the ARS algorithm implements adaptive resource allocation for each service-slice instance. The integration of these algorithms constitutes the overall MAARS framework, which leverages a multiagent system based on the actor–critic structure to efficiently manage network resources, thereby optimizing overall performance. It aims to maximize resource utilization while minimizing resource conflicts between network slices and services, enabling tailored resource allocation and scheduling based on the characteristics of each instance. Thus, it offers a robust solution that not only enhances network performance but improves overall system quality (QoS) through efficient resource utilization.

4.2. MADDPG Algorithm

MADDPG is a multiagent RL algorithm that aims to learn the policy (π_i) and Q-function (Q_i) of each agent. The algorithm employs centralized training and decentralized execution, wherein a central controller trains an agent-network group based on global environmental information, whereas the agents select and execute actions based on local environmental information [29,30]. Decentralized execution mitigates the problem of the joint-action space growing exponentially as the number of agents increases; therefore, traditional single-agent RL algorithms are unsuitable for this task.

At each step of the MADDPG algorithm, agents first obtain local observations $o_j(t)$ of the environment, select actions $a_j(t)$ based on $o_j(t)$, where j represents the agent index, and

execute the selected actions to receive a reward *r(t)*. After executing the selected actions, the environment is updated, and the agents receive new local observations $o_j(t+1)$ in the next step. The central controller collects the new observations of all agents and stores the experience ($s(t)$, $a(t)$, $r(t)$, $s(t+1)$) in a replay buffer, where $s(t)$ and $s(t+1)$ denote the vectors containing the observations of all agents at time steps t and $t+1$, respectively. A batch of experiences is then sampled from the replay buffer to train the networks for all agents in the MADDPG algorithm [31].

MADDPG uses a set of agent networks and a central critic network to learn policy and Q-function $Q_{i(o,a_i)}$ of each agent as follows:

Agent networks: These networks employ a DDPG structure, which includes both the evaluation and target networks, and are used to learn the policies π_i and Q-functions of each agent. The input for the agent networks comprises the current observation $o_j(t)$, and they output the action $a_j(t)$ or Q-value $Q_{j(o,a_j)}$.

Central critic network: This network takes the observations and actions of all agents as inputs and outputs the Q-value for the Q-function of each agent. Using this information obtained from all agents, the central critic network computes the Q-values that allow each agent to learn a better policy.

Similar to the DQN, MADDPG trains the parameters of all agent networks by minimizing a loss function, which is defined as follows:

$$L(\theta_i) = \mathbb{E}_{(o,a,r,o')}\left[(y_i - Q_i(o, a_1, \ldots, a_N; \theta_i)^2\right], \tag{16}$$

$$y_i = r_i + \gamma Q_i'(o', a_1', \ldots, a_N'; \theta_i'), \tag{17}$$

where Q_i' and θ_i' denote the target network and its parameters, respectively, θ_i denotes the parameters of the evaluation network, and a_N' is the action selected by the target network.

The MEC-network environment can be reflected by the current number of user computational tasks, network state, and computing resources available for each network-slice entity. The virtual controller of each network-slice entity can only observe the local environment and manage its network-slice entity. Thus, the network model can be considered a decentralized execution model. However, each network-slice entity is interdependent. Because the total number of network and computing resources is limited across all ENs and all M logical access networks share the same resources, ECS must be conducted using a centralized training mode to obtain an optimal slicing policy.

Thus, the MADDPG algorithm, which employs centralized learning and decentralized execution, is well-suited for solving SP1 to achieve an optimal policy. Compared to the independent Q-learning (IQL) algorithm, MADDPG can enhance the relationship between the policy π_i and Q-function $Q_{i(o,a_i)}$ of each agent [32]. The IQL algorithm involves independent learning by each agent, which can lead to poor performance in environments where interaction is important. By contrast, the MADDPG algorithm considers agent interactions through a central critic, thereby enhancing performance.

MADDPG-Based ECS Algorithm

The MADDPG-based ECS algorithm efficiently manages core-to-edge resource allocation in NS environments, optimizing resource utilization to maximize task success rates. By leveraging MADDPG, the ECS layer effectively addresses the challenges of coordinating resource distribution across the network hierarchy, enabling a dynamic and adaptive response to changing demands.

To accelerate learning and further enhance performance, we employ MAML for initialization. This significantly reduces the convergence time of MADDPG, allowing the ECS algorithm to adapt quickly to evolving network conditions and ensuring consistent performance. Additionally, its integration with the SAC-based ARS layer ensures that the outputs from ECS seamlessly guide real-time resource slicing, creating a cohesive and efficient optimization loop that spans the entire network. This two-tiered approach ensures coordinated resource management from the core to the edge.

Compared to traditional single-agent or static optimization methods, our MADDPG-based ECS algorithm offers significant advantages. It provides improved resource efficiency, dynamic adaptability to complex and evolving demands, and higher task-completion rates. Moreover, it offers a robust and scalable solution for optimizing resource allocation for MEC NS. The agents, states, actions, and reward functions are defined as follows:

1. Agent: The agent acts as a virtual controller for each network-slice entity periodically performing ECS. Each agent monitors the resource status of its respective slice instance and makes optimal resource-allocation decisions.
2. State: The state is represented by a set of local observations of all agents. Each observation includes the resource and task status obtained from the network-slice entity and is defined as follows:

$$o_m(t) = \{SE_{w,m,n}(t), RN_{w,m,n}(t),\ SR_{w,m,n}(t),\ PU_{w,m,n}(t) | n = 1,\ 2,\ \ldots,\ N_m, w = 1,\ 2, \ldots,\ W\}, \tag{18}$$

where $SE_{w,m,n}(t)$ represents the number of available NRU resources at the n-th BS of the slice entity and $RN_{w,m,n}(t)$ indicates the number of available CRU resources at the n-th EN. Additionally, $SR_{w,m,n}(t)$ denotes the task-completion rate at the n-th edge node, $PU_{l,m,n}(t)$ represents the number of user computational tasks per time unit, and N_m indicates the number of BS/EN. Based on the local observations of each agent, the state is defined as follows:

$$s(t) = \{o_m(t) | m = 1,\ 2,\ \ldots,\ M\}. \tag{19}$$

3. Action: Each agent performs ECS within its respective network-slice entity. Specifically, it can choose to increase, decrease, or maintain the number of NRUs or CRUs in each BS. The action space of each agent is expressed as follows:

$$a_m(t) = \{SV_{w,m,n}(t), SW_{w,m,n}(t) | n = 1, \ldots, N_m, w = 1, \ldots, W\}, \tag{20}$$

where $SV_{w,m,n}(t) = 1$ and $SW_{w,m,n}(t) = 1$ represent increases in the NRU and CRU resources, respectively, at the n-th edge server of the network-slice entity. Conversely, $SV_{w,m,n}(t) = -1$ and $SW_{w,m,n}(t) = -1$ indicate decreases in the NRU and CRU resources, respectively, whereas $SV_{l,m,n}(t) = 0$ and $SW_{l,m,n}(t) = 0$ signify no changes in resources. The number of resource blocks that can increase or decrease at each step is fixed. The joint-action vector of the MADDPG algorithm is defined as follows:

$$a(t) = \{a_m(t) | m = 1,\ 2,\ \ldots,\ M\}. \tag{21}$$

4. Reward Function: Because the goal of the algorithm is to maximize the long-term weighted average of the task-completion rate, a penalty-based reward function was designed. If the task-completion processing rate falls below a certain threshold, a penalty is applied to reduce the reward. This reward structure encourages higher performance by penalizing agents that lead to poor performance, thereby driving them to achieve better results. The reward function is defined as

$$\gamma_m(t) = \begin{cases} \omega_2 \cdot \chi_m(t) + \overline{\omega}_2 \ \ if\ \chi_m(t) \geq \chi_{threshold} \\ \omega_2 \cdot \chi_m(t) + \overline{\omega}_2 - \Gamma\ if\ \chi_m(t) < \chi_{threshold} \end{cases} \tag{22}$$

where, $\omega_2 > 0$ is a coefficient that determines the reward slope based on the task-completion rate and $\overline{\omega}_2$ sets the offset value of the reward. Additionally, $\chi_{threshold}$ is the threshold of the task-completion rate and $\Gamma > 0$ is the penalty applied when the performance falls below the threshold.

Figure 2 illustrates the structure of the MADDPG-based MAARS algorithm, where multiple agents interact with network-slice entities to optimize resource allocation. Each agent utilizes actor and critic networks to collaboratively manage resources and ensure efficient task processing across network slices. Algorithm 1 summarizes the main steps of

the MADDPG-based ECS algorithm, which represents an approach wherein each agent acts independently but cooperates with other agents through centralized learning to efficiently allocate resources in an NS environment.

Algorithm 1: MADDPG-based ECS

1: Initialize policy networks π_i and Q-functions Q_i for all agents
2: Initialize target networks π'_i and Q'_i with the same parameters
3: Initialize replay buffer R
4: **for** episode = 1 to NP_{epis}:
5: Initialize random exploration process
6: Initialize state s_0
7: **for** step = 1 to NQ_{step} :
8: for each agent i:
9: Observe local state $o_i(t)$ and select action $a_i(t) = \pi_i(o_i(t)) + noise$
10: Execute joint-action $a(t) = \{a_1(t), \ldots, a_M(t)\}$
11: Observe reward $r_m(t)$ and next state $s(t+1)$
12: Store experience $(s(t), a(t), r_m(t), s(t+1)$ in R
13: Sample mini-batch from R, for each agent i:
14: Compute target action $a_i(t+1)$ and target Q-value y_i
15: Update Q_i by minimizing L(θ_i)
16: Update π_i using policy gradient
17: Soft update target networks θ'_i and θ'_{Q_i}
18: Decay exploration noise for next episode

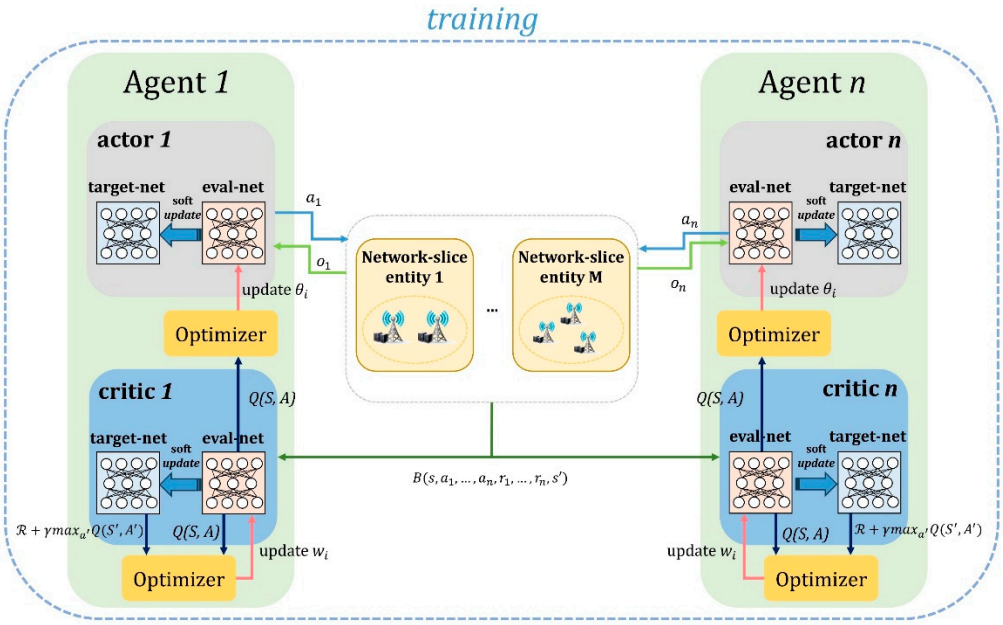

Figure 2. Illustration of the MADDPG-based ECS algorithm.

The policy network and Q-function for each agent are initialized, and the same parameters are set for the target network. A random exploration process is initialized at the start of each episode. In each step, the agent observes the local state and selects an action. After executing the selected action, the agent receives a reward, observes the next state and the reward, and stores the experience in the replay buffer. Next, the Q-function is updated using a mini-batch sampled from the replay buffer, and the policy network is updated based on the policy gradient. Finally, the target network is soft-updated, and the exploration noise is reduced for the next episode. This algorithm aims to maximize the efficiency of network-resource allocation and optimize the task-completion rate.

4.3. MAML-Based Initialization Algorithm

For a given weight vector, the MADDPG-based ECS algorithm periodically performs ECS to learn the optimal policy aligned with that vector. However, when the weight vector changes, relearning the optimal policy can be time-consuming [33–35]. To address this, the MAML-based initialization algorithm is introduced, which obtains a meta-policy a set of optimal initialization parameters for the agent, mixture, and hyper-networks employed in the MADDPG-based algorithm. This enables faster adaptation to new weight vectors, thereby significantly reducing the time required to re-learn the optimal policy.

4.3.1. Meta-Policy of MAML-Based Initialization Algorithm

The meta-policy provides an initial optimized state that can quickly adapt to various objectives with different weight vectors. This serves as an optimal initial policy that can be effectively trained for diverse objectives [36]. The MAML-based initialization algorithm is model-agnostic and can learn a meta-policy that swiftly adapts to different goals.

By providing a robust initialization, MAML accelerates the convergence of the MADDPG-based ECS algorithm, ensuring better performance, even under dynamic network conditions. The meta-policy allows agents to start from a well-optimized state, reducing the iterations needed to achieve stable policies. This is crucial in MEC networks, where rapid adaptation is essential for handling fluctuating resource demands and maintaining service quality.

In this algorithm, the meta-policy parameters are updated based on the optimal policy parameters for each objective. This algorithm comprises inner and outer loops. In the inner loop, the policy parameters are trained for a specific objective to determine the optimal policy, and gradient descent is employed to minimize the loss function of the MADDPG-based ECS algorithm. In the outer loop, the gradients of each optimal policy parameter are calculated to update the meta-policy parameters.

Integrating MAML into MADDPG enhances the algorithm's ability to handle diverse and dynamic network environments with improved efficiency and adaptability. This capability is crucial for optimizing resource allocation and enhancing the overall performance of MEC networks.

4.3.2. MAML-Based Initialization Algorithm for Policy Optimization

The MAML-based initialization algorithm comprises the following three main stages:

- Adaptation: A set of weight vectors $A = [a_1, a_2, \ldots, a_i]$ is given, where i represents the total number of elements and each vector contains a series of weighting coefficients. SP1 determines the optimal policy for each weight vector. To achieve this, the policy is periodically updated using each weight vector. In the inner loop, the policy parameters ϕ_i are optimized for each weight vector a_i through gradient descent as follows:

$$\phi_i' = \phi_i - \omega_3 \nabla_{\phi_i} \mathscr{L}_{a_i}(\phi_i), \tag{23}$$

  where ϕ_i' and $\overline{\omega}_3$ denote the updated parameters and learning rate, respectively. Through this process, the optimal policy parameters ϕ_i^* for each weight vector are obtained as follows:

$$\phi_i^* = arg \min_{\phi_i} \mathscr{L}_{a_i}(\phi_i). \tag{24}$$

- Meta-policy Training: In this stage, the optimal policy parameters ϕ_i^* obtained in the previous stage are used together to learn the optimal meta-policy parameters. In the outer loop, the optimal policy parameters for various weight vectors from the inner loop are used to update the meta-policy parameters as follows:

$$\Theta = \Theta - \iota \sum_i \nabla_\Theta \mathscr{L}_{a_i}(\phi_i^*), \tag{25}$$

where ι represents the metalearning rate. Through this process, Θ of the meta-policy are optimized.

- Fine-tuning: To achieve the objective of SP1 with a new weight vector α_{new}, the meta-policy parameters are used as the initial parameters for the agent, mixture, and hyper-networks in the MADDPG-based algorithm. Subsequently, by iteratively fine-tuning the initial policy using this algorithm, the optimal resource-allocation policy π_{ECS} aligned with α_{new} can be obtained. This process aims to maximize ECS performance by efficiently allocating resources within the network-slice instance while maintaining or enhancing the completion rate of user computational tasks. This stage includes updating the policy parameters by minimizing the loss function as follows:

$$\Theta'_{new} = \Theta - \mu \sum_{i} \Theta \mathscr{L}_{a_i}(\phi), \tag{26}$$

where μ is the learning rate during the fine-tuning stage. The MAML-based initialization algorithm offers several benefits. First, it allows for rapid learning of an optimal policy for new weight vectors through the meta-policy, thereby enabling quick adaptation to various environmental changes. Second, it minimizes overall training time by reducing the time required to re-learn the policy using initialized parameters. This is particularly beneficial in scenarios that require repetitive learning. Third, it can create a policy that works effectively across different weight vectors, thereby increasing system generalizability. These advantages help improve overall system performance and maximize efficiency. Algorithm 2 summarizes the key steps of the MADDPG-based ECS algorithm, which aims to learn a set of initialized parameters that can be used to quickly determine the optimal policy for various weight vectors.

Algorithm 2: MAML-based Initialization

1: Initialize meta-policy parameters Θ
2: **for** iteration = 1 to $num_{iterations}$:
3: Sample a batch of weight vectors $\{a_1, a_2, \ldots, a_B\}$ from A
4: **for** each weight vector a_i perform:
5: Initialize policy parameters φ_i with Θ
6: Update φ_i using gradient descent on $\mathscr{L}_{a_i}(\phi_i^*)$ over K steps
7: Store the optimal policy parameters φ_{i^*}
8: **end for**
9: Compute meta-policy update: $\Delta\Theta = -\eta \sum_i \nabla_\Theta \mathscr{L}_{a_i}(\phi_i^*)$
10: Update meta-policy parameters: $\Theta = \Theta + \Delta\Theta$
11: **end for**
12: Fine-tune policy parameters Θ using new weight vector α_{new}

In the first step of the algorithm, the initial meta-policy parameters are set. These parameters are trained to provide optimal initialization across multiple weight vectors. Subsequently, the meta learning loop begins, wherein the meta-policy is learned over various iterations. In each iteration, various weight vectors are sampled, and the optimal policy parameters are learned based on them. For each batch of sampled weight vectors, the policy parameters are initialized based on a meta-policy. In the inner loop, these parameters are optimized through gradient descent. The optimized policy parameters are then stored and used to update the meta-policy.

Finally, when a new weight vector is provided, the algorithm quickly fine-tunes the optimal policy based on the parameters initialized by the meta-policy. This process continuously enhances the meta-policy, enabling rapid adaptation to new environments.

4.4. SAC-Based Resource-Allocation Management (RAM) Algorithm

The SAC-based RAM algorithm plays a critical role in efficiently managing network resources at the edge, providing the flexibility and scalability required to meet diverse user

demands. A policy-based optimization algorithm known for its high sample efficiency and stability [37], SAC is particularly well-suited for the dynamic resource management challenges inherent in MEC networks. Within the MAARS framework, SAC complements the MADDPG-based ECS algorithm by focusing on real-time, fine-grained resource slicing at the network edge, guided by the outputs of ECS.

This dual-layer approach combines the global coordination capabilities of MADDPG with the precise local optimization of SAC. MADDPG handles the core-to-edge resource allocation, while SAC refines this allocation at the edge, providing a comprehensive solution to the challenges of resource management in MEC networks. This integration ensures seamless and efficient resource allocation across the entire network hierarchy. Compared to traditional methods, the SAC-based RAM algorithm offers significant advantages. It not only improves resource utilization and task success rates but dynamically adapts to the ever-changing demands of the network. The combination of MADDPG and SAC delivers a robust and scalable solution for MEC network slicing, ensuring superior performance and adaptability even in the most complex and dynamic environments.

The learning process of this algorithm comprises multiple iterations. During training, experience replay and target networks are used to increase stability. Experience replay improves sample efficiency and reduces data correlation by using a buffer to store the experiences of each agent [38]. The target network is periodically updated, which contributes to the stability of the learning process.

- Agent: The agent is responsible for making resource-allocation decisions and corresponds to network nodes such as the BS or edge servers. Each agent allocates resources based on the network state and learns resource-management policies while providing services to users.
- State: The state comprises a set of parameters that represent the current network conditions. This state information is essential for an agent to understand the environment and optimize resources. The state variables are defined as follows:

$$State(t) = \left\{ Resource_{Usage}(t),\ Queue_{Length}(t),\ Demand_{User}, Delay_{Max} \right\}, \tag{27}$$

  where $Resource_{Usage}(t)$ represents the number of resources currently in use and indicates the ratio of allocated resources to the total resources available in the network. $Queue_{Length}(t)$ reflects the queue length, which indicates the number of tasks waiting to be processed in the network. Additionally, $Demand_{User}$ refers to the number of resources that must be allocated to each user in the network and $Delay_{Max}$ represents the maximum allowable delay to deliver each service to the users. These state variables play a critical role in network-resource management by providing essential information for the agent to make optimal resource-allocation decisions.
- Action: The action of an agent involves determining the number of resources that must be allocated for each user request based on the current state and is defined as follows:

$$Action(t) = \{a_i(t) | i \in \{1, 2, \ldots, Count_{User}(t)\}\}, \tag{28}$$

  where $a_i(t)$ represents the number of resources allocated to user i at time t. $\in \{1, 2, \ldots, Count_{User}(t)\}$ represents the total number of users connected to the network. The agent makes resource-allocation decisions for each user. Therefore, $Action(t)$ is an action vector that represents the resource allocation for all users, thereby playing a critical role in efficient network-resource management and meeting user demands.
- Reward: The reward function evaluates the resource-allocation efficiency and guides the agent to learn better policies. It is defined as the difference between the actual and target resource usage, as follows:

$$reward(t) = \sum_{i=1}^{Count_{User}(t)} \left(\frac{\vartheta_{max} - \vartheta_{actual,\,i}(t)}{\vartheta_{max}} \cdot \mathbb{I}(\vartheta_{actual,\,i}(t) \leq \vartheta_{max}) - \mathbb{I}(\vartheta_{actual,\,i}(t) > \vartheta_{max}) \right). \tag{29}$$

This function evaluates the resource-allocation efficiency at a specific time t. If the actual transmission delay $\vartheta_{actual,i}$ for each user i, is within the maximum allowable delay ϑ_{max}, the reward is maintained as $\frac{\vartheta_{max}-\vartheta_{actual,i}(t)}{\vartheta_{max}}$. This encourages efficient resource allocation by granting higher rewards for shorter delays. Conversely, if $\vartheta_{actual,i}(t) \leq \vartheta_{max}$, a penalty is applied to that agent. The total reward is calculated by summing these values across all users, thereby indicating the overall efficiency of network-resource allocation. This reward structure aims to optimize resource allocation and guide the agent to minimize delays during learning.

Learning Process of SAC-Based RAM Algorithm

The SAC-based RAM algorithm enhances learning stability by using experience replay and target networks. Experience replay includes a buffer that stores user experiences, enhances sample efficiency, and minimizes data correlation. The target network is only updated periodically to further enhance the stability of the learning process. The pseudocode of the learning process of this algorithm is presented in Algorithm 3.

Algorithm 3: SAC-based RAM

Initialize actor-network parameters θ_p and critic-network parameters ϕ
Initialize target-network parameters φ' with φ
for episode = 1 to NP_{epis} perform:
for t = 1 to NP_{epis} perform:
Observe current state s_t
Select action a_t using policy $\pi(\theta_p \mid s_t)$
Execute action a_t to receive reward r_t and next state s_{t+1}
Store (s_t, a_t, r_t, s_{t+1}) in replay buffer
Sample a random mini-batch from replay buffer
Compute temporal-difference error δ_t:
$\delta_t = r_t + \gamma \times Q_{\varphi'}(s_{t+1}, \pi(\theta_p \mid s_{t+1})) - Q_{\varphi}(s_t, a_t)$
Update critic network by minimizing:
$L(\varphi) = (1/2) \times \sum(\delta_t{}^2)$
Update actor network using:
$J_{SAC}(\theta_p) = \mathbb{E}_t\,[\min(\psi_t(\theta_p), \mathrm{clip}(\psi_t\,(\theta_p), 1-\epsilon, 1+\epsilon)) \times \delta_t]$
Update target network ϕ':
if t % target_update_interval == 0:
$\varphi' \longleftarrow \varphi$

This algorithm focuses on maximizing the efficiency of network-resource allocation and learning an optimal resource-management policy. It enables stable and flexible resource allocation under various network conditions and can adapt to network performance fluctuations. The SAC-based approach is particularly effective in high-dimensional continuous action spaces and offers robust performance for solving complex resource-allocation problems. Moreover, the SAC algorithm provides high sample efficiency, allowing the agent to quickly learn optimal resource-allocation policies, even with limited training data. Consequently, it reduces the complexity of network-resource management while ensuring optimal resource utilization and improving the QoS.

4.5. Computational Complexity

The computational complexity of the proposed MAARS framework is determined by the operations in Algorithms 1, 2, and 3, each contributing to the overall efficiency and scalability of the system. Algorithm 1, the MADDPG-based ECS algorithm, primarily involves nested loops over tasks and resources. This results in a time complexity of $O(N \times M)$, where N represents the tasks and M represents the resources, with a space complexity of $O(N)$ for task states. Algorithm 2, the MAML-based initialization algorithm, incorporates matrix operations for optimizing the meta-policy parameters. This leads

to a time complexity of $O(N \times M + M^2)$ and a space complexity of $O(N + M^2)$ for the resource matrices. Algorithm 3, the SAC-based RAM algorithm, handles RL-based policy updates for resource allocation. The time complexity of this algorithm is $O(N \times M)$, where N represents the number of slices and M represents the number of iterations in the learning process, and a space complexity of $O(N + P)$, where P represents the policy parameters. By combining the complexities of these individual algorithms, the overall MAARS framework achieves a time complexity of $O(N \times M + M^2)$ and a space complexity of $(N + M^2 + P)$. This analysis demonstrates that the proposed framework is designed for scalability and efficiency, making it suitable for deployment in large-scale MEC networks with numerous tasks, resources, and network slices.

4.6. Performance Evaluation

The performances of the proposed algorithms were validated through simulations. First, to evaluate the SAC-based RAM algorithm, we compared its performance with that of the random resource-allocation [39], proportional fairness (PF) [40], and centralized resource-allocation (CRA) [41] algorithms. The random resource-allocation algorithm randomly allocates resources to user tasks, whereas the PF algorithm allocates them fairly based on both resource availability and user demands. By contrast, the CRA algorithm manages all network resources centrally and aims to achieve optimal resource distribution.

Next, the proposed MAARS algorithm was evaluated by comparing its efficiency and performance with those of heuristic-based, DQN-based, and advantage actor–critic (A2C)-based resource-allocation algorithms. The heuristic-based algorithm allocates resources based on predefined rules, the DQN-based algorithm allocates them based on the data size and computation requirements of a task, and the A2C-based algorithm conducts periodic resource allocation using the A2C algorithm.

To further assess the performance of the MAARS framework, we conducted a comprehensive analysis of the impact of MAML. By leveraging MAML to initialize the ECS neural networks with pretrained meta-policy parameters, we observed a significant reduction in the number of training iterations required for convergence. This advanced initialization strategy enabled the ECS component to adapt more effectively to dynamic network conditions and fluctuating resource demands, ultimately enhancing the overall performance and scalability of the framework.

The evaluation metrics included the success rate for user tasks within the service-slice entities and the weighted average of the success rates across all network-slice entities. To comprehensively evaluate the contribution of MAML, we analyzed additional metrics, such as convergence time and resource-allocation adaptability. The results demonstrated faster convergence and more stable resource utilization, particularly under dynamic network conditions, highlighting the effectiveness of MAML in enhancing the adaptability and efficiency of the framework. The simulations were implemented using Python (version 3.12.4) and PyTorch (version 2.4), assuming that user-task arrivals followed a Poisson process. The system configuration used for the simulations included an Intel Core i7-9700K CPU (Santa Clara, CA, USA) running at 3.60 GHz, 32 GB of DDR4 RAM, and an NVIDIA RTX 2080 Ti GPU (Santa Clara, CA, USA). The key simulation parameters are listed in Table 2.

Table 2. Hyperparameters employed in the simulations.

Parameter	Value
Number of ENs (N)	10
Number of network-slice entities (M)	4
Transmission power $p_k(t)$	23 dBM
Noise power (N_0)	−95 dBm
CPU frequency	2.5 GHz
Bandwidth	5 MHz

Table 2. *Cont.*

Parameter	Value
Batch size	64
Learning rate (α)	0.001
Discount factor (γ)	0.99
Experience pool size	1000

5. Results

5.1. SAC-Based RAM Algorithm

Figure 3 compares the task-completion ratios at the BS within the service-slice entity and the task-arrival rates for the SAC-based RAM and the adaptive and random resource-allocation algorithms. As the task-arrival rate increased, the completion rates of all algorithms gradually decreased. This is because as more computational tasks arrive at the BS under a higher arrival rate, processing delays occur, leading to a higher percentage of tasks failing to meet the maximum allowable service delay. However, the proposed SAC-based RAM algorithm demonstrates a better task-processing rate than the other two algorithms.

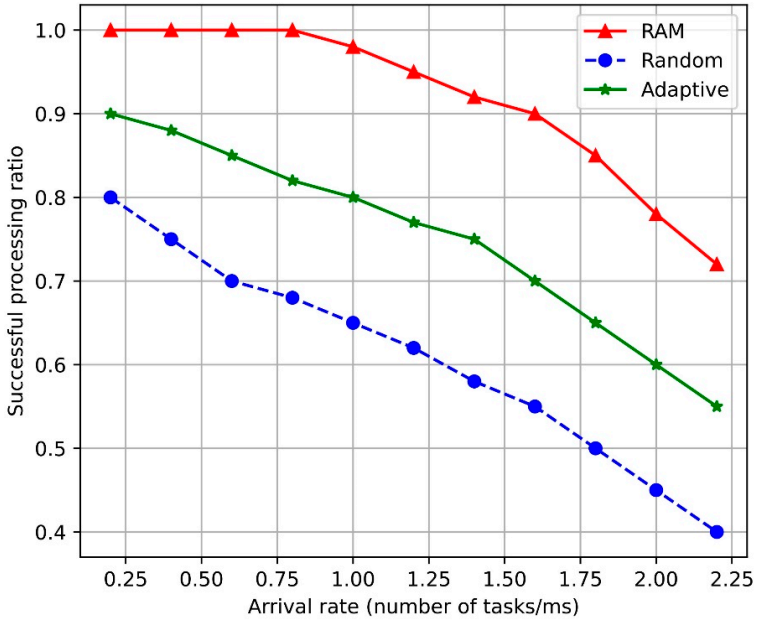

Figure 3. Task-completion ratio vs. arrival rate. RAM: resource-allocation management.

The SAC-based RAM algorithm exhibits a higher task-completion ratio because it dynamically allocates both network and computing resources by simultaneously considering the task demand and current resource state of the BS. This allows it to efficiently allocate resources, even under heavy loads, thereby minimizing task-processing delays. Specifically, by continuously monitoring network conditions in real-time and optimally readjusting resource availability, unnecessary wait times are minimized, and the task-completion rate is increased. Thus, the SAC-based RAM algorithm provides stable performance even if the network load increases, thereby maximizing resource efficiency.

Figure 4 compares the task-completion ratios of the SAC-based RAM, adaptive, and random algorithms at the BS based on the CPU frequency. As the CPU frequency increases, the task-completion ratios of all algorithms initially increase before stabilizing. This trend occurs because a higher CPU frequency increases the computing resources that can be allocated to tasks, thereby enhancing the likelihood of processing tasks within the maximum allowable service delay. Although the task-completion ratios of all algorithms increased, that of the SAC-based RAM was the highest.

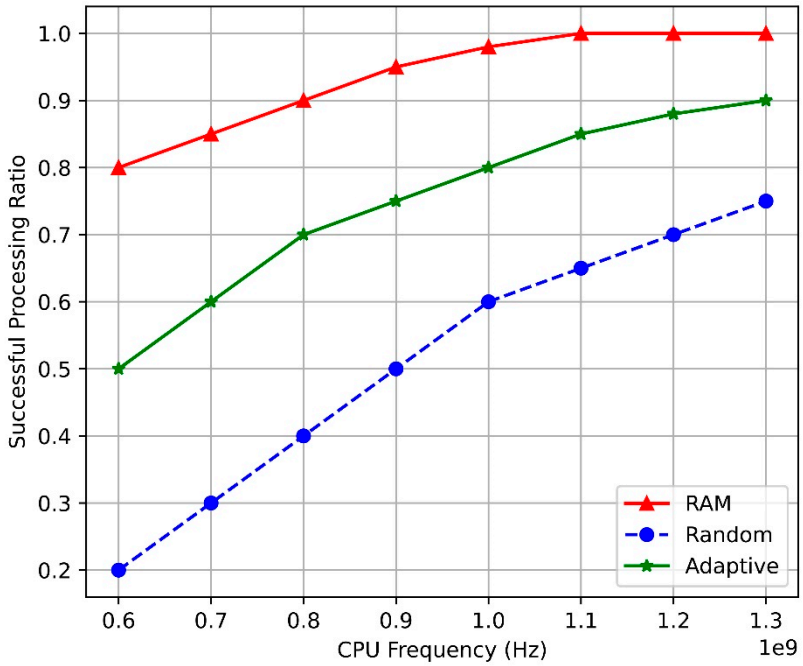

Figure 4. Task-completion ratio vs. CPU frequency.

This is because the design of the proposed RAM algorithm is based on the SAC framework, which enables dynamic resource allocation and CPU utilization optimization, ensuring that appropriate computing resources are allocated to each task in real-time. The SAC-based RAM algorithm maintains resource efficiency by considering both CPU status and user-task demands, even when the CPU frequency increases. This highlights that merely increasing the CPU frequency has limitations in improving the task-completion ratio, underscoring the importance of resource management and allocation optimization.

Figure 5 compares the task-completion ratios of the SAC-based RAM, adaptive, and random algorithms at the BS based on bandwidth. As the bandwidth increased, the task-completion ratios of all algorithms initially increased before stabilizing. This pattern occurs because a larger bandwidth allows more spectrum resources to be allocated to the tasks, increasing the likelihood of processing them within the maximum allowable delay. However, once sufficient bandwidth is allocated, the processing capacity becomes saturated, and additional bandwidth increases have a lower impact on performance.

Nevertheless, the SAC-based RAM algorithm maintains a higher task-completion ratio than the other algorithms through real-time resource optimization and efficient allocation. This is because the RL-based approach dynamically analyzes network conditions and task demands to allocate resources appropriately.

Therefore, the proposed SAC-based RAM algorithm demonstrates superior performance to the other algorithms under key resource conditions, such as arrival rate, CPU frequency, and bandwidth, and maintains a higher task-completion ratio even under resource-overload conditions. Owing to its dynamic optimization and real-time adaptability, the SAC-based RAM algorithm can play a crucial role in more complex network environments in the future. Consequently, it can be essential for maximizing network-resource utilization and maintaining QoS.

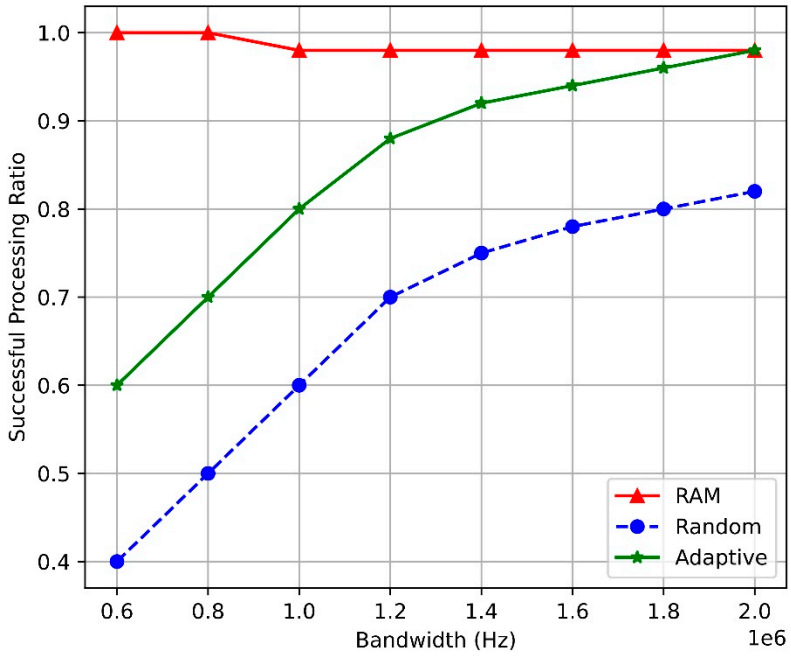

Figure 5. Task-completion ratio vs. bandwidth.

5.2. MADDPG-Based MAARS Algorithm

Figure 6 illustrates the relationship between the reward values and the number of iterations for the MADDPG-based MAARS algorithm under conditions of both ample and scarce network and computing resources. As the number of iterations increases, the reward values converge in both environments, indicating that the algorithm progressively learns to allocate resources more efficiently and reaches a stable performance level. In environments with ample resources, the MAARS algorithm achieves higher reward values, reflecting its effectiveness in optimizing resource allocation through efficient utilization. The MADDPG-based distributed resource-management technique allows the MAARS algorithm to cooperatively allocate resources across network slices, ensuring that the needs of each slice are met optimally.

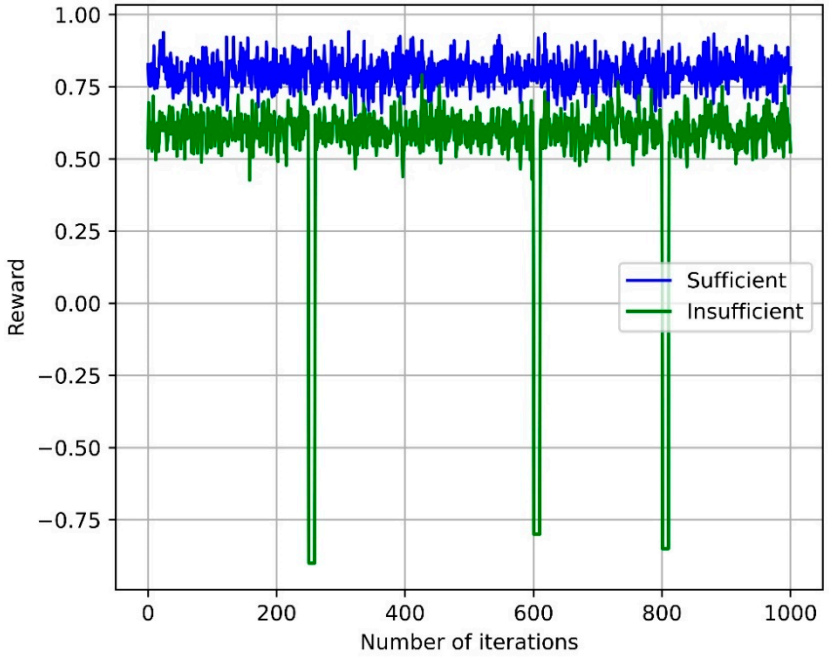

Figure 6. Reward values vs. number of iterations.

Moreover, even in resource-scarce situations, MAARS maintains its performance efficiency by adapting to network-state changes and implementing real-time resource allocation. This helps prevent excessive usage or shortages of resources. Thus, the MAARS algorithm demonstrated stable and high performance across various environments.

Figure 7 illustrates the relationship between the task-completion ratios and task-arrival rate for the MAARS, DQN–PPO, PF, CRA, and heuristic-based resource-allocation algorithms. Evidently, as the arrival rate increases, the task-completion ratios of all algorithms decrease owing to increased wait times and service delays. However, both the MAARS and DQN–PPO algorithms maintain higher task-completion ratios than the PF, CRA, and heuristic algorithms. Specifically, MAARS leverages the MADDPG-based collaborative resource-allocation technique to dynamically adjust the resources for each slice, thereby maximizing performance.

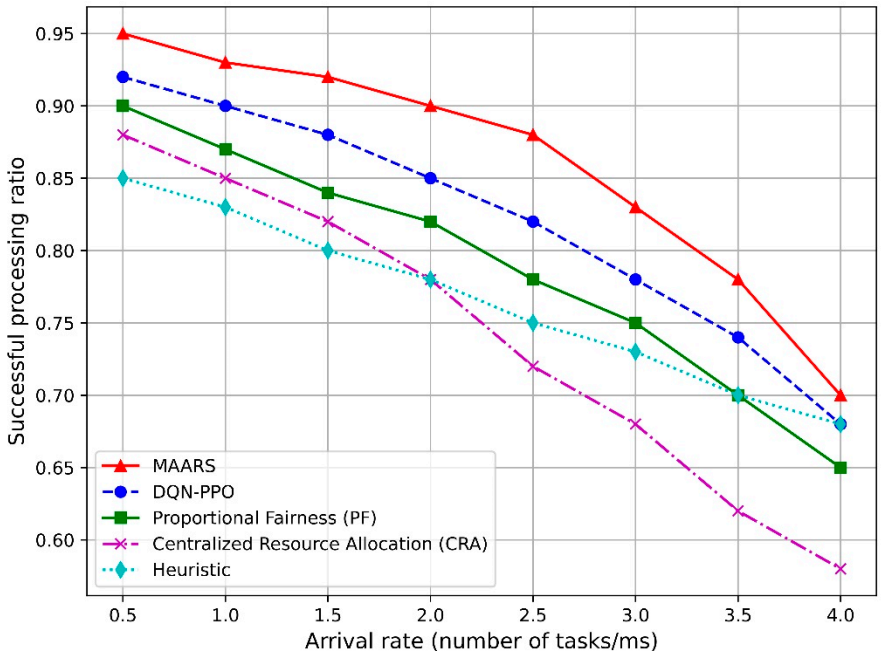

Figure 7. Task-completion ratio vs. arrival rate.

Figure 8 illustrates the task-completion ratio as a function of the CPU frequency. Evidently, as the CPU frequency increases, the completion ratio of all algorithms increases because a higher CPU frequency enhances the task-processing speed. However, the MAARS algorithm exhibits the most significant performance improvement owing to its collaborative resource-allocation feature, which efficiently distributes resources based on the requirements of each network slice. Additionally, although the performance of DQN–PPO also increases with the CPU frequency, it does not reach the level of MAARS. By contrast, the PF, CRA, and heuristic algorithms show relatively lower performance owing to limitations in their resource-allocation optimization strategies, whereas MAARS maximizes efficiency in this scenario.

Figure 9 illustrates the task-completion ratio as a function of the bandwidth. As the bandwidth increases, the task-completion ratio of all algorithms improves before stabilizing after a certain bandwidth threshold. MAARS and DQN–PPO exhibit higher performances than the PF, CRA, and heuristic algorithms because MAARS offers more efficient and collaborative resource management and allocation. At the same time, DQN–PPO benefits from combining the strengths of both deep Q-networks and proximal policy optimization, leading to more effective decision-making in dynamic environments.

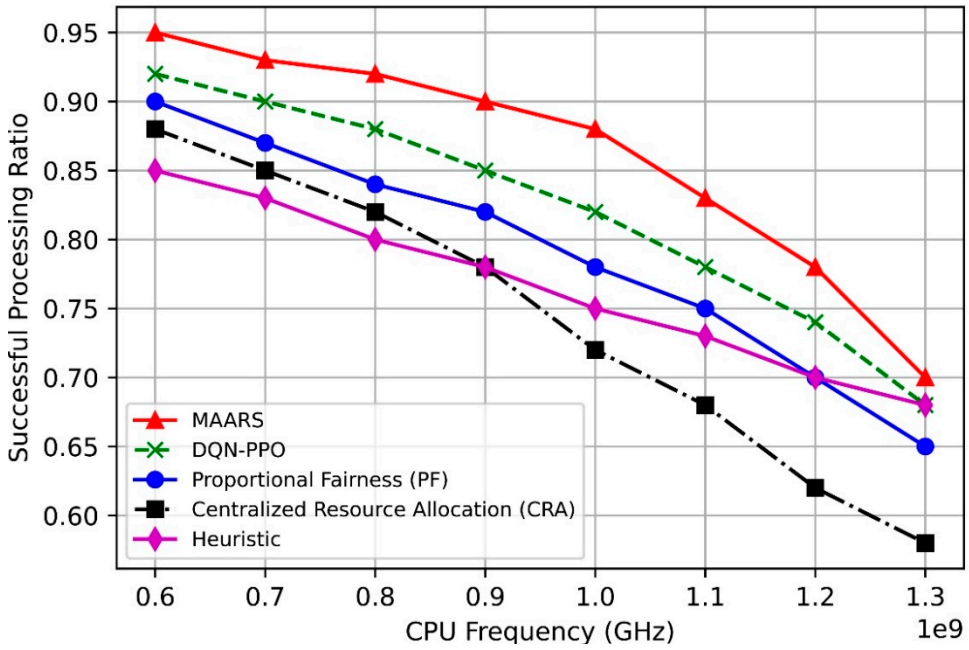

Figure 8. Task-completion ratio vs. CPU frequency.

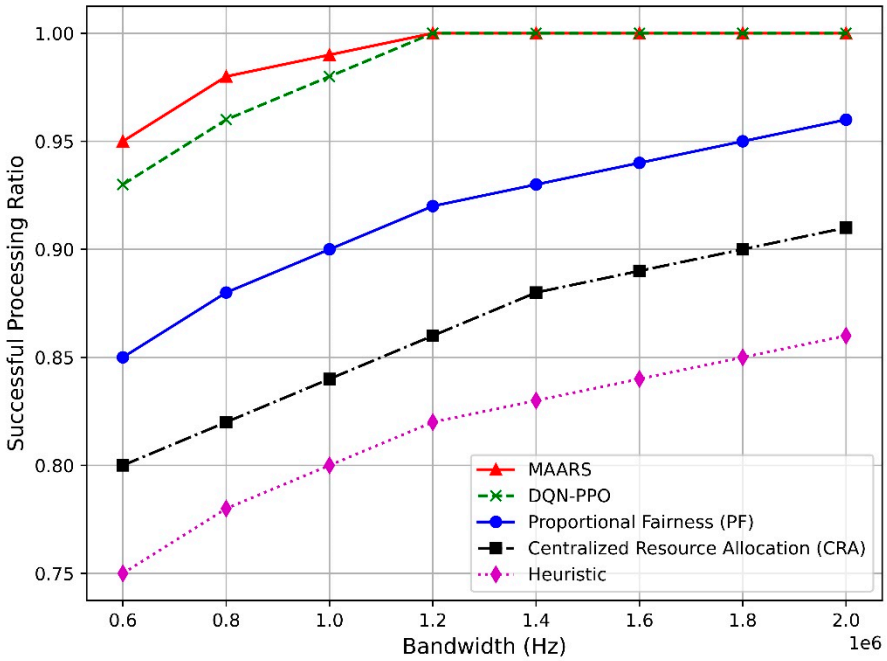

Figure 9. Task-completion ratio vs. bandwidth.

MAARS is designed to reflect the real-time requirements of network slices, ensure timely resource allocation, and maximize resource utilization without wastage. By contrast, DQN–PPO manages resources independently for each slice, which limits its optimization compared with MAARS. Moreover, PF focuses on fairness, which restricts performance optimization, whereas CRA struggles to adapt quickly to real-time changes owing to its centralized approach. Additionally, the heuristic-based algorithm uses fixed resource-allocation rules, failing to adapt to complex network demands. Therefore, MAARS efficiently utilizes resources even as the bandwidth increases, maintaining a higher task-completion ratio than the other algorithms.

Figure 10 shows the utility-function values for three network-slice instances obtained using the MADDPG-based MAARS algorithm with respect to various weight vectors. Each

weight vector represents a specific objective function, with the "initial" value indicating the utility before NS. The MAARS algorithm adopts a collaborative distributed resource-management approach for each network slice, leading to more efficient resource allocation. Assigning higher weights to specific network slices increases their utility-function values, reflecting the capability of MAARS to intelligently adjust resource-allocation priorities among network slices. By reflecting the real-time demands of each slice, MAARS can allocate resources in a timely manner and maximize performance. Compared to traditional algorithms, such as DQN–PPO, MAARS demonstrates superior performance owing to its collaborative resource allocation, which minimizes resource wastage and ensures that optimal performance is maintained.

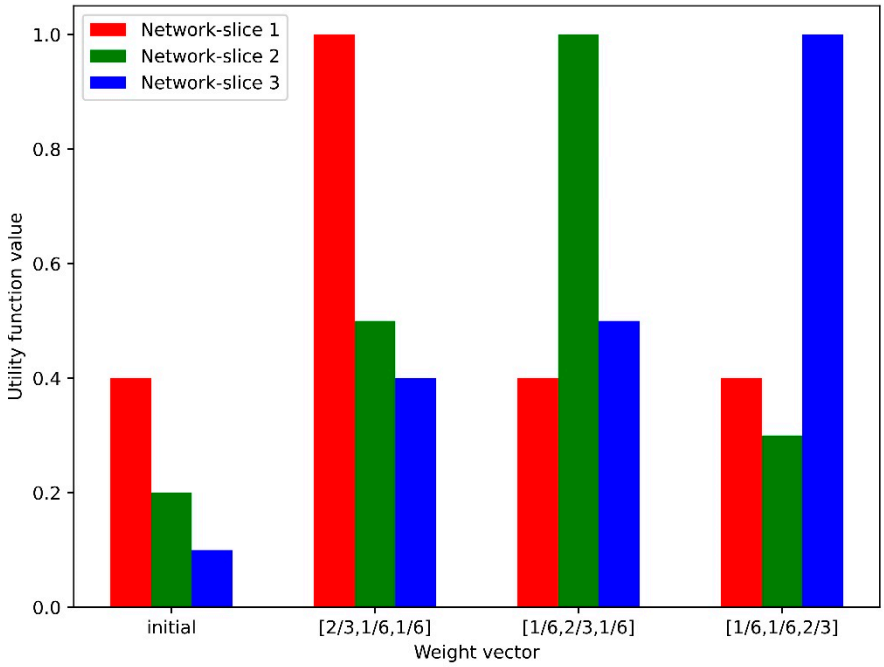

Figure 10. Utility-function values vs. weight vectors.

Figure 11 illustrates the loss ratios as the number of iterations increases for various resource-allocation algorithms, including MAARS, DQN–PPO, PF, CRA, and heuristic-based. As the number of iterations increases, the loss ratio of MAARS decreases; thus, it outperforms all other algorithms in terms of convergence speed and efficiency. DQN–PPO also demonstrates significant loss reduction, but lags behind MAARS. PF and CRA show more moderate loss reduction, whereas the heuristic algorithm exhibits the highest loss ratio. MAARS achieves superior performance owing to its collaborative and adaptive resource management, which ensures efficient task processing and resource allocation across network slices. By contrast, DQN–PPO, although effective, allocates resources independently for each slice, which limits its overall optimization performance. Furthermore, PF focuses on fairness, which constrains its ability to optimize performance, while CRA struggles to adapt quickly to dynamic network conditions owing to its centralized approach. Finally, the heuristic-based algorithm relies on fixed rules and, therefore, cannot effectively adapt to the complexity of real-time network demands. Consequently, MAARS consistently demonstrated superior loss-reduction performance over time than the other algorithms.

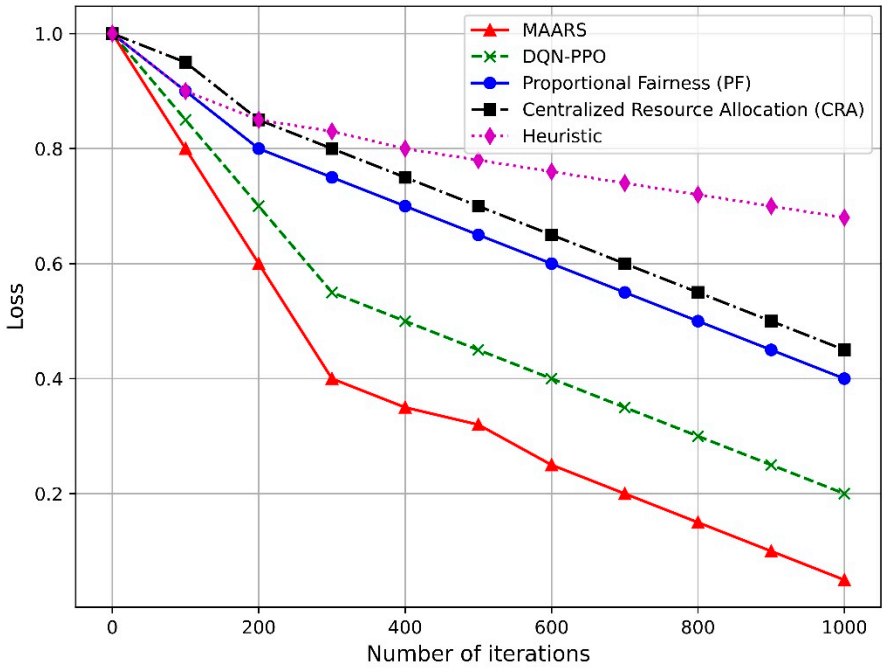

Figure 11. Loss ratio vs. number of iterations.

6. Discussion

This study introduced the MAARS algorithm to effectively address the challenges of resource allocation and NS in MEC networks. MAARS leverages collaborative resource allocation to achieve superior performance than existing algorithms, dynamically adapting to network changes in real-time.

The simulation results validated the efficiency of the MADDPG-based resource management technique, which efficiently allocates resources based on the real-time demands of each network slice. The integration of MADDPG and SAC creates a powerful dual-layer optimization framework. MADDPG manages global resource coordination, while SAC provides real-time adaptability at the network edge. This approach enhances scalability and operational efficiency across a wide range of network conditions. Future research should explore the potential of the MAARS algorithm in even more complex and heterogeneous network environments, further expanding its applicability for resource allocation in challenging scenarios.

7. Conclusions

This study presented a novel two-stage resource slicing and allocation model to address the critical challenges of NS and resource allocation in MEC networks. This model, formulated as a long-term optimization problem, aims to maximize the completion rate of user-computing tasks while effectively meeting the diverse service requirements of individual network slices. To achieve this, we decomposed resource management into two distinct yet interconnected algorithms: ECS and ARS, integrated within the proposed MAARS algorithm. To enhance the efficiency of the MADDPG-based ECS algorithm, we employed a MAML initialization method. This approach provides optimal initial policies for the ECS agents, significantly reducing learning time and enabling rapid adaptation to changing network conditions. The integration of MADDPG and SAC establishes a powerful dual-layer optimization framework, wherein MADDPG efficiently manages global resource slicing across the network, while SAC ensures real-time adaptability and fine-grained resource allocation at the network edge. This synergistic approach delivers superior scalability and efficiency, outperforming existing methods in managing network slices under dynamic conditions. Thus, the proposed MAARS algorithm offers a robust and effective solution for optimizing resource allocation and network performance in complex MEC environments.

Its adaptability to varying network conditions makes it a promising approach for managing the evolving demands of next-generation networks. Future research should focus on further enhancing the performance of MAARS and extending its applicability to more complex and heterogeneous network scenarios.

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

Funding: This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II201373, Artificial Intelligence Graduate School Program (Hanyang University)).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available upon request from the corresponding author, I.J. The data are not publicly available because they contain information that can compromise the privacy of the research participants.

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

Appendix A

This appendix details the delay factors considered for the performance evaluations and provides the equation used to calculate the task-completion rate. The total service delay for a task comprises several critical components. These include the transmission-preparation delay, which represents the time required to execute the processes, such as encoding and packaging, and to prepare the data for transmission; transmission delay, which refers to the time required for the data to travel across the network; and waiting delay, which is the time a task spends in queue before being processed owing to factors such as network congestion or high demand for computing resources. Additionally, computation-preparation delay is the time required to set up the computational environment, including resource allocation and initialization. Finally, computation-waiting and processing delays encompass the time a task waits for available computing resources and the actual computation time once the processing begins, respectively.

The total service delay is calculated by summing several delay components, including the transmission-preparation, transmission, waiting, computation-preparation, and waiting and processing delays during computation. This comprehensive metric accounts for the total time spent handling a task, from start to finish. Thus, it reflects the total time required to process a task and includes both the network and computational delays.

The system performance was evaluated based on the task-completion rate, which indicates the effectiveness of task processing within the allowable delay limits. A task is deemed to be successfully processed if its total service delay is less than or equal to the maximum allowable response time T_{max}.

The success rate $\rho_{w,m,n}(t)$ is defined as the ratio of completed tasks to the total number of tasks, adjusted by a function of the system performance relative to a threshold. It is calculated as follows:

$$\rho_{w,m,n}(t) = \frac{\mathcal{U}_{success,w,m,n}(t)}{\mathcal{U}_{total,w,m,n}(t)} \cdot \left(\frac{1}{1 + \exp(-o \cdot (\mathcal{R}_{w,m,n}(t) - \mathcal{R}_{th}} \right)$$

$\mathcal{U}_{success,w,m,n}(t)$ represents the number of tasks successfully processed at time t, where a successfully processed task is that whose total service delay does not exceed T_{max}. Additionally, $\mathcal{U}_{total,w,m,n}(t)$ denotes the total number of tasks processed up to time t, including those processed successfully and those that failed to process. $\mathcal{R}_{w,m,n}(t)$ represents the current

response time or system performance at time t, reflecting the observed task-processing performance. $\mathscr{R}_{th}$ is the performance threshold beyond which the system performance is deemed suboptimal, whereas o controls the sensitivity of the success rate to deviations from this threshold; a higher value amplifies the effect of performance deviations on the success rate.

This equation integrates the success rate with an adjustment factor based on the response time. $\frac{1}{1+\exp(-o\cdot(\mathscr{R}_{w,m,n}(t)-\mathscr{R}_{th}}$ is a sigmoid function that moderates the impact of response-time variations on the success rate. As $\mathscr{R}_{w,m,n}(t)$ approaches or exceeds the value of $\mathscr{R}_{th}$, the adjustment factor reduces the success rate, reflecting poorer performance. This comprehensive approach ensures a thorough evaluation of the system's ability to satisfy the response-time requirements and provides valuable insights into the effectiveness of task-processing and resource-management strategies.

Abbreviations

The following abbreviations are used in this manuscript:

AR	Augmented reality
A2C	Advantage actor critic
ARS	Autonomous resource slicing
BS	Base station
BRB	Bandwidth resource block
CRA	Centralized resource allocation
CRU	Computing resource unit
DDPG	Deep deterministic policy gradient
DQN	Deep Q-network
ECS	Efficient core-to-edge slicing
ES	Edge server
IoT	Internet of Things
IQL	Independent Q-learning
MADDPG	Multi-agent deep deterministic policy gradient
MAML	Model–agent–critic resource scheduling
MEC	Multi-access edge computing
NFV	Network function virtualization
NS	Network slicing
NSS	Network-slice segment
NRU	Network resource unit
PF	Proportional fairness
PPO	Proximal policy optimization
PRS	Processing resource block
SAC	Soft actor–critic
SDN	Software defined networking
SDU	Smart device user
VR	Virtual reality

References

1. Han, X.; Zhong, Y.; Zhang, L. An efficient and robust integrated geospatial object detection framework for high spatial resolution remote sensing imagery. *Remote Sens.* **2017**, *9*, 666. [CrossRef]
2. Blaschke, T.; Hay, G.J.; Weng, Q.; Resch, B. Collective sensing: Integrating geospatial technologies to understand urban systems—An overview. *Remote Sens.* **2011**, *3*, 1743–1776. [CrossRef]
3. Leyva-Mayorga, I.; Martinez-Gost, M.; Moretti, M.; Pérez-Neira, A.; Vázquez, M.Á.; Popovski, P.; Soret, B. Satellite edge computing for real-time and very-high resolution earth observation. *IEEE Trans. Commun.* **2023**, *71*, 6180–6194. [CrossRef]
4. Ullo, S.L.; Sinha, G.R. Advances in IoT and smart sensors for remote sensing and agriculture applications. *Remote Sens.* **2021**, *13*, 2585. [CrossRef]
5. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
6. Zhang, Y.; Di, B.; Zheng, Z.; Lin, J.; Song, L. Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 2565–2578. [CrossRef]

7. He, Y.; Zhai, D.; Huang, F.; Wang, D.; Tang, X.; Zhang, R. Joint task offloading, resource allocation, and security assurance for mobile edge computing-enabled UAV-assisted VANETs. *Remote Sens.* **2021**, *13*, 1547. [CrossRef]
8. Alameddine, H.A.; Sharafeddine, S.; Sebbah, S.; Ayoubi, S.; Assi, C. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. *IEEE J. Select. Areas Commun.* **2019**, *37*, 668–682. [CrossRef]
9. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on multi-access edge computing for internet of things realization. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2961–2991. [CrossRef]
10. Wang, J.; Hu, J.; Min, G.; Zhan, W.; Ni, Q.; Georgalas, N. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Commun. Mag.* **2019**, *57*, 64–69. [CrossRef]
11. Ngene, C.E.; Thakur, P.; Singh, G. Power allocation strategies for 6G communication in VL-NOMA systems: An overview. *Smart Sci.* **2023**, *11*, 475–518. [CrossRef]
12. Luong, N.C.; Hoang, D.T.; Gong, S.; Niyato, D.; Wang, P.; Liang, Y.C.; Kim, D.I. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 3133–3174. [CrossRef]
13. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
15. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; PMLR: New York, NY, USA, 2014; pp. 387–395.
16. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
17. Tran, T.X.; Pompili, D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. Veh. Technol.* **2018**, *68*, 856–868. [CrossRef]
18. Cevallos Moreno, J.F.; Sattler, R.; Caulier Cisterna, R.P.; Ricciardi Celsi, L.; Sánchez Rodríguez, A.; Mecella, M. Online service function chain deployment for live-streaming in virtualized content delivery networks: A deep reinforcement learning approach. *Future Internet* **2021**, *13*, 278. [CrossRef]
19. Huang, H.; Ye, Q.; Zhou, Y. Deadline-aware task offloading with partially observable deep reinforcement learning for multi-access edge computing. *IEEE Trans. Netw. Sci. Eng.* **2021**, *9*, 3870–3885. [CrossRef]
20. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Emerg. Top. Comput.* **2019**, *9*, 1529–1541. [CrossRef]
21. Nduwayezu, M.; Pham, Q.V.; Hwang, W.J. Online computation offloading in NOMA-based multi-access edge computing: A deep reinforcement learning approach. *IEEE Access* **2020**, *8*, 99098–99109. [CrossRef]
22. Ning, Z.; Yang, Y.; Wang, X.; Guo, L.; Gao, X.; Guo, S.; Wang, G. Dynamic computation offloading and server deployment for UAV-enabled multi-access edge computing. *IEEE Trans. Mob. Comput.* **2021**, *22*, 2628–2644. [CrossRef]
23. Shah, S.D.A.; Gregory, M.A.; Li, S. Toward network slicing enabled edge computing: A cloud-native approach for slice mobility. *IEEE Internet Things J.* **2023**, *11*, 2684–2700. [CrossRef]
24. Khan, A.A.; Abolhasan, M.; Ni, W.; Lipman, J.; Jamalipour, A. An end-to-end (E2E) network slicing framework for 5G vehicular ad-hoc networks. *IEEE Trans. Veh. Technol.* **2021**, *70*, 7103–7112. [CrossRef]
25. Wu, W.; Chen, N.; Zhou, C.; Li, M.; Shen, X.; Zhuang, W.; Li, X. Dynamic RAN slicing for service-oriented vehicular networks via constrained learning. *IEEE J. Select. Areas Commun.* **2020**, *39*, 2076–2089. [CrossRef]
26. Seah, W.K.G.; Lee, C.H.; Lin, Y.D.; Lai, Y.C. Combined communication and computing resource scheduling in sliced 5G multi-access edge computing systems. *IEEE Trans. Veh. Technol.* **2021**, *71*, 3144–3154. [CrossRef]
27. Jin, J.; Li, R.; Yang, X.; Jin, M.; Hu, F. A network slicing algorithm for cloud-edge collaboration hybrid computing in 5G and beyond networks. *Comput. Electr. Eng.* **2023**, *109*, 108750. [CrossRef]
28. Chiang, Y.; Hsu, C.H.; Chen, G.H.; Wei, H.Y. Deep Q-learning-based dynamic network slicing and task offloading in edge network. *IEEE Trans. Netw. Serv. Manage.* **2022**, *20*, 369–384. [CrossRef]
29. Zhang, H.; Du, Y.; Zhao, S.; Yuan, Y.; Gao, Q. VN-MADDPG: A variable-noise-based multi-agent reinforcement learning algorithm for autonomous vehicles at unsignalized intersections. *Electronics* **2024**, *13*, 3180. [CrossRef]
30. Li, S.; Wu, Y.; Cui, X.; Dong, H.; Fang, F.; Russell, S. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. *Proc. AAAI Conf. Artif. Intell.* **2019**, *33*, 4213–4220. [CrossRef]
31. Long, Q.; Zhou, Z.; Gupta, A.; Fang, F.; Wu, Y.; Wang, X. Evolutionary population curriculum for scaling multi-agent reinforcement learning. *arXiv* **2020**, arXiv:2003.10423. [CrossRef]
32. Kostrikov, I.; Nair, A.; Levine, S. Offline reinforcement learning with implicit q-learning. *arXiv* **2021**, arXiv:2110.06169. [CrossRef]
33. Mao, H.; Zhang, Z.; Xiao, Z.; Gong, Z. Modelling the dynamic joint policy of teammates with attention multi-agent DDPG. *arXiv* **2018**, arXiv:1811.07029. [CrossRef]
34. Jeon, S.; Lee, H.; Kaliappan, V.K.; Nguyen, T.A.; Jo, H.; Cho, H.; Min, D. Multiagent reinforcement learning based on fusion-multiactor-attention-critic for multiple-unmanned-aerial-vehicle navigation control. *Energies* **2022**, *15*, 7426. [CrossRef]
35. Zhang, L.; Li, J.; Yang, Q.; Xu, C.; Zhao, F. MADDPG-based deployment algorithm for 5G network slicing. *Electronics* **2024**, *13*, 3189. [CrossRef]

36. Cheng, Z.; Min, M.; Liwang, M.; Huang, L.; Gao, Z. Multiagent DDPG-based joint task partitioning and power control in fog computing networks. *IEEE Internet Things J.* **2021**, *9*, 104–116. [CrossRef]
37. Peng, B.; Rashid, T.; Schroeder de Witt, C.; Kamienny, P.A.; Torr, P.; Böhmer, W.; Whiteson, S. FACMAC: Factored multi-agent centralised policy gradients. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 12208–12221.
38. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; PMLR, 2018; pp. 1861–1870.
39. Czumaj, A.; Stemann, V. Randomized allocation processes. *Random Struct. Algorithms* **2001**, *18*, 297–331. [CrossRef]
40. Ramjee, T.B.L.E.L.R.; Bu, T.; Li, L.E. Generalized proportional fair scheduling in third generation wireless data networks. In Proceedings of the IEEE Infocom, Barcelona, Spain, 23–29 April 2006; pp. 1–12.
41. Leconte, M.; Paschos, G.S.; Mertikopoulos, P.; Kozat, U.C. A resource allocation framework for network slicing. In Proceedings of the IEEE INFOCOM IEEE Conference on Computer Communications, Honolulu, HI, USA, 15–19 April 2018; pp. 2177–2185. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.