Doctoral Thesis

Network Resource Management Using Multi-Agent Deep Reinforcement Learning

Akito SUZUKI

Graduate School of Informatics, Kyoto University

September 2023

Preface

The future evolution of networks will require various new communication technologies. These technologies offer many advantages, including flexibility, scalability, and cost savings. Network functions virtualization (NFV) is one of the key technologies of future networks. NFV enables telecommunications service providers (TSPs) to offer various network services by flexibly combining multiple virtual network functions (VNFs), which can improve resource utilization efficiency by sharing physical resources between VNFs. To provide such services, a TSP should allocate virtual network (VN) requests represented by virtual nodes as VNFs and virtual links connecting virtual nodes. A significant research challenge in NFV is finding an optimal VN allocation considering limited network and server resources. Edge computing (EC) is also one of the key technologies of future networks. EC can provide computing capability at the edge servers close to end devices (EDs). Diverse applications are generally offloaded and processed in edge servers or cloud servers because of the limitation of the computation resources of EDs. Such offloaded applications' tasks consist of the demand for computing and communication with various characteristics. A significant research challenge in EC is finding efficient task offloading to edge and cloud servers considering various task characteristics and limited network and server resources.

In common with both technologies, network resource management must be optimized to maximize resource utilization efficiency in a network with limited physical resources. The optimal management refers to the allocation of VNs and tasks that maximizes the objective function (e.g., resource utilization) while satisfying constraints (e.g., resource capacity). The performance of allocation algorithms is critical for future network management, as it determines the overall network's resource utilization efficiency. This thesis studies

four specific problems about network resource management using multi-agent deep reinforcement learning (MADRL). Each problem corresponds to resource-integrated control in NFV, dynamic VN allocation, and task offloading for multi-cloud-edge networks, respectively. This thesis focused on reinforcement learning (RL) as a solution to the resource management problem because it can quickly calculate a near-optimal resource allocation by learning the relationship between input network resource patterns and output resource allocation in advance.

Firstly, this thesis proposes an extendable network-resource-integrated control method based on the coordinated control architecture in NFV, which consists of multiple pre-specified control algorithms and a single coordination algorithm between the control algorithms. The key idea for extendability is modularization that divides a whole system into standardized functional elements and reduces the interdependence among the elements. This work first prepares and solves each control algorithm for each control metric and then interactively improves the results using the proposed coordination algorithm. This work also proposes an efficient coordination algorithm on the basis of RL. The learning makes it possible to learn the strategy for how to find better allocations efficiently from past exploration steps. The proposed method requires more iterations than the combined approach, but it achieves extendability through the coordinated control architecture. Finally, this work evaluated the effectiveness of the proposed method through simulations.

Secondly, this thesis proposes a dynamic VN allocation method based on safe multi-agent deep reinforcement learning (Safe-MADRL). This method can quickly optimize network resources even while network demands are drastically changing by using the deep reinforcement learning (DRL) in advance. This work develops two techniques to be used with the proposed method; safety-considerations and multi-agent. The proposed safety-considerations technique reduces the degree of constraint violations, such as network congestion and server overload, and the proposed multi-agent technique improves the scalability of VN allocation by dividing demands into groups and assigning each group's allocation to each agent. As a result of a simulation evaluation, Safe-MADRL can calculate effective allocation within one second that doubles the link utilization efficiency without any constraint violations compared to the

static VN allocation method.

Thirdly, this thesis proposes a dynamic VN allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method also can quickly optimize network resources even while network demands are drastically changing by using DRL in advance. The key idea is to use a multi-agent technique for an RL-based dynamic VN allocation method, which can reduce the number of candidate actions per agent and can improve the performance for VN allocation. Moreover, a cooperation technique improves the efficiency of VN allocation. From results of a simulation evaluation, Coop-MADRL can calculate effective allocation within one second, which reduces the maximum server and link utilization and drastically reduces the constraint violations compared with that of the static VN allocation method. Furthermore, this work revealed that the learning with various mixed traffic models could achieve a high generalization performance for all traffic patterns.

Fourthly, this thesis proposes a task-offloading method for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. This work introduces a cooperative multi-agent technique through centralized training and decentralized execution, improving task-offloading efficiency. Simulations revealed that the proposed method can minimize network utilization and task latency while minimizing constraint violations in less than one millisecond in various network topologies. It also shows that cooperative learning improves the efficiency of task offloading. This work demonstrated that the proposed method has generalization performance for various task types by pre-training with many resource-consuming tasks.

This thesis is organized as follows. Chapter 1 introduces the background of network resource management in future networks. Chapter 2 investigates the related works in literature. Chapter 3 describes the extendable resource-integrated control method in NFV and an efficient algorithm for the proposed method using RL. Chapter 4 describes the Safe-MADRL-based dynamic VN allocation method. Chapter 5 describes the Coop-MADRL-based dynamic VN allocation method. Chapter 6 describes the Coop-MADRL-based task-offloading method. Finally, Chapter 7 concludes this thesis.

Acknowledgements

I would like to express my gratitude to everyone who has supported and assisted me throughout the completion of my doctoral thesis.

First and foremost, I would like to express my deepest gratitude to my doctoral advisor, Professor Eiji Oki of Kyoto University, for his guidance and support. I am truly grateful for the opportunity to pursue my doctoral research in his laboratory. Without his valuable advice and encouragement, my studies would never have been accepted in high-level journals, and this thesis would never have been completed.

Next, I would like to express my deep gratitude to my previous supervisors, Professor Ryoichi Kawahara of Toyo University, Professor Keisuke Ishibashi of International Christian University, and Professor Takanobu Watanabe of Waseda University. Their guidance and encouragement have greatly influenced my growth as a researcher and shaped me into the researcher I am today.

I would also like to express my gratitude to Professor Hiroshi Harada and Professor Takayuki Ito of Kyoto University for being members of my judging committee and providing valuable feedback to improve this thesis.

Furthermore, I would like to acknowledge all the current and past colleagues at NTT Laboratory for their cooperation and inspiration. In particular, I would like to express my gratitude to Dr. Masataka Masuda, Dr. Shigeaki Harada, Mr. Masahiro Kobayashi, Mr. Yousuke Takahashi, and Dr. Yoichi Matsuo for their contributions and discussions throughout my research.

Lastly, this thesis is dedicated to my beloved wife, Miyabi, my precious son, Takayoshi, and my supportive parents. I would like to express my special gratitude to my wife for her dedicated support. Her strong encouragement and deep understanding have enabled me to concentrate on my studies. Without her presence, none of these accomplishments would have been possible.

Contents

List of Figures

List of Tables

Notations

Notation	Description
a_t^{agent}	Action of each agent at step t (agent $\in$ ia $\cup \boldsymbol{G}$)
$a_t^e \in \mathcal{A}^e$	Action for agent g_e at step t
$a_{k,t} \in \mathcal{A}_k$	Action of k^{th} agent at step t
$\mathcal{A} := \{\mathcal{A}^e\}$	Action sets for all agents ($\mathcal{A}^e$: Action space)
$\boldsymbol{A}^e$	Solution of control engine e
$\boldsymbol{a}_t := \{a_t^e\}$	All actions at step t
$B_k^{\text{up}}, B_k^{\text{down}}$	Upload/Download traffic demand of k^{th} task
$\boldsymbol{B}_t := \left\{b_t^i\right\}$	Traffic demands of i^{th} VN at step t
$c \in \boldsymbol{C} \subset \boldsymbol{N}$	Cloud
c_{ij}, c_{ij}^L, c_{ij}^{link}	Link capacity of link (i, j)
c_i, c_i^S, c_i^{server}	Server capacity of server i
c_i^{ids}	i^{th} IDS capacity
$\boldsymbol{C}_k$	Computing demand of k^{th} task
$\boldsymbol{C}$	Client set
$\mathcal{D} := \{\boldsymbol{D}_k\}$	Task set ($k \in \mathcal{K}$, $\mathcal{K}$: Total tasks)
$\boldsymbol{D_t} := \left\{d_{ijt}\right\}$	Traffic demands from node i to node j at step t
$\boldsymbol{D}_t := \left\{d_t^i\right\}$	VM demands of i^{th} VN at step t
$\boldsymbol{D}_t := \left\{d_{i,t}\right\}$	Traffic demands of i^{th} VN at step t
$e \in E$	Episode (E: Total episodes)
$e \in \boldsymbol{E} \subset \boldsymbol{N}$	Edge
$\boldsymbol{E} := \{e\}$	Control engine set
$G(\boldsymbol{S}, \boldsymbol{L})$	Network graph ($\boldsymbol{S}$: server set, $\boldsymbol{L}$: link set)
$\boldsymbol{G} := \{g\}$	Control agent set

Notation	Description
$\mathcal{G} := \{g_e\}$	Agent set
$\mathcal{G}^c := \left\{g_k^c\right\}$	Constraint agent sets ($1 \leq k \leq M$)
$\mathcal{G}^o := \left\{g_k^o\right\}$	Objective agent sets ($1 \leq k \leq M$)
$h^i \in \boldsymbol{h}$	Observation-action history
$\boldsymbol{h}$	Observation-action history of all agents
$\boldsymbol{I}$	IDS set
$\mathcal{L}_p$	Link set along with path p
link $(i, j) \in \boldsymbol{L}$	Link from node i to node j
$\mathcal{M}$	Replay memory
$\boldsymbol{M}_t := \left\{m_t^{pq}\right\}$	Traffic matrix from node p to node q
$n \in \boldsymbol{N}$	Node
$N_{\text{cli}}, N_{\text{vm}}, N_{\text{ids}}$	Number of clients, VMs, and IDSs
N_{server}	Number of servers
N_{VN}	Number of VNs
$o_t^e \in O^e$	Observation for agent g_e at step t
$\boldsymbol{O} := \{O^e\}$	Observation sets for all agents
$\boldsymbol{o}_t := \{o_t^e\}$	All observations at step t
$P(\boldsymbol{N}, \boldsymbol{L})$	Physical Network graph
P_t	Penalty Function of VM migration
$\boldsymbol{P} := \{p_i\}$	i^{th} User placement
$\mathcal{P}_k^{\text{up}}$, $\mathcal{P}_k^{\text{down}}$	Upload and download path set of k^{th} task
$Q\left(s_t, a_{k,t}\right)$	Action-value function for s_t and $a_{k,t}$
$Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$	Joint action-value function for all agent
$Q_e\left(o_t^e, a_t^n\right)$	Action-value function for agent g_e
r_t^{agent}	Reward of each agent at step t (agent $\in$ ia $\cup \boldsymbol{G}$)
R^{total}	Total reliability
r_i^{node}	i^{th} node reliability
r_{ij}^{link}	Link reliability of link (i, j)
r_p	Traffic-splitting ratio of path p
r_t	Reward for agent g_e at step t
$\boldsymbol{R}_t^L := \left\{r_{ij,t}^L\right\}$	(i, j) residual link resources at step t
$\boldsymbol{R}_t^S := \left\{r_{i,t}^S\right\}$	i^{th} residual server resources at step t

Notation	Description
$s \in \boldsymbol{S}$	Server
s_t^{agent}	State of each agent at step t (agent $\in$ ia $\cup$ $\boldsymbol{G}$)
$s_t \in \mathcal{S}$	State at step t ($\mathcal{S}$: State space)
$t \in \boldsymbol{T}$	Time-step ($\boldsymbol{T}$: Total time-steps)
$t^{\text{sim}} \in \boldsymbol{T}^{\text{sim}}$	Time-step for simulator ($\boldsymbol{T}^{\text{sim}}$: Total time steps)
t_i^{VN}	OD Traffic demands for i^{th} VN
t^g	Number of iterations of control agent ($g \in \boldsymbol{G}$)
$\boldsymbol{T}^g$	Total exploration steps of control agent ($g \in \boldsymbol{G}$)
t_k	Acceptance time of k^{th} task
$\boldsymbol{T} := \{t_{pq}\}$	Traffic matrix from server p to server q
$\boldsymbol{T}^{\text{node}} := \{t_{pq}\}$	Traffic from node p to node q
$\boldsymbol{T}^{\text{vm}} := \left\{t_{ij}^{\text{vm}}\right\}$	Traffic from VM i to VM j
$U_{\max}^{\text{link}}$	Maximum link utilization
$U_{\max}^{\text{server}}$	Maximum server utilization
$u_{ij,t}^L$	(i, j) link utilization at step t
$u_{i,t}^N$	i^{th} node utilization at step t
$u_{i,t}^S$	i^{th} server utilization at step t
$U_t^L = \max_{ij}\left(u_{ij,t}^L\right)$	Maximum link utilization at step t
$U_t^N = \max_i\left(u_{i,t}^N\right)$	Maximum node utilization at step t
$U_t^S = \max_i\left(u_{i,t}^S\right)$	Maximum server utilization at step t
$v \in \boldsymbol{V}$	VM ($\boldsymbol{V}$: VM set)
v_i^N	i^{th} node-processing capability
$\boldsymbol{V_t} := \{v_t^e\}$	Evaluation values of control engine e at step t
$\boldsymbol{V}_t := \{v_{i,t}\}$	VM demands of i^{th} VN at step t
$w \in \boldsymbol{W}$	User
w_{ij}^L	(i, j) link capacity
w_i^N	i^{th} node capacity
w_i^{vm}, w_j^{ids}	i^{th} VM size, j^{th} IDS size
w_c	Weighting parameter of constraint agents $\mathcal{G}^c$
x_{ij}^{pq}	Proportion of passed t_{pq} on link (i, j)
$\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$	Proportion of passed traffic matrix on link (i, j)

Notation	Description
$\boldsymbol{Y} := \{y_{kn}\}$	Task allocation (task k, node n)
$\boldsymbol{Y}_t := \{y_{ij,t}\}$	VM allocation at step t (VM i, server j)
$\boldsymbol{Z} := \{z_{ij}\}$	User/device placement (user i, node j)
α_{ij}^{L}	Propagation latency of link (i, j)
β_k	Task type of k^{th} task
λ	Weighting parameter of objective function
$\mathbb{I}_{k,t}$	Binary variable indicating executing tasks
$\tau_k^{\max}$	Maximum permissible latency of k^{th} task
τ_k^{N}, τ_k^{RTT}	Node latency and RTT latency of k^{th} task
τ_p	Latency coefficient of path p
$\boldsymbol{\tau}_t := \{\tau_t^{pq}\}$	Traffic matrix from node p to node q
$\boldsymbol{\theta} := \{\theta^e\}$	Coefficients of control engine e
$\boldsymbol{\Xi}^{\text{cli}} := \{\xi_{ij}^{\text{cli}}\}$	Client node placement (i^{th} client, j^{th} node)
$\boldsymbol{\Xi}^{\text{ids}} := \{\xi_{ij}^{\text{ids}}\}$	IDS allocation (i^{th} IDS, j^{th} server)
$\boldsymbol{\Xi}^{\text{vm}} := \{\xi_{ij}^{\text{vm}}\}$	VM allocation (i^{th} VM, j^{th} server)

Abbreviations

Abbreviation	Description
ARMA	autoregressive moving average
BS	base station
CC	cloud computing
CEV	comprehensive evaluation value
CNF	cloud-native network function
CPU	central processing unit
CSI	channel state information
DC	data-center
Dec-POMDP	decentralized partially observable Markov decision process
DNN	deep neural network
DQN	deep Q-network
DRL	deep reinforcement learning
DRQN	deep recurrent Q-Network
EC	edge computing
ED	end device
ES	exhaustive search
FIFO	first-in-first-out
GA	greedy algorithm
GLPK	GNU Linear Programming Kit
GRU	gated recurrent unit
HA	heuristic algorithm
HRA	hybrid reward architecture
IDS	intrusion detection system

Abbreviation	Description
ILP	integer linear programming
IoT	Internet of Things
IQL	independent Q-learning
LP	linear programming
LSTM	recurrent long short-term memory
MADRL	multi-agent deep reinforcement learning
MARL	multi-agent reinforcement learning
MINLP	mixed-integer non-linear programming
MIP	mixed integer programming
MOGA	multi-objective genetic algorithm
NFV	network functions virtualization
NLP	non-linear Programming
OD	origin-destination
OS	operating system
RA	random algorithm
RAM	random access memory
RL	reinforcement learning
RNN	recurrent neural network
RTT	round-trip time
SA	static allocation
SARIMA	seasonal autoregressive integrated moving average
SDN	Software Defined Networking
SFC	service function chaining
TSP	telecommunications service provider
VDN	value decomposition network
VM	virtual machine
VN	virtual network
VNE	virtual network embedding
VNF	virtual network function
WAN	wide-area network

Chapter 1

Introduction

1.1 Network resource management in future networks

1.1.1 Network resource management in NFV

Network functions virtualization (NFV) [1–3] is one of the key technologies of future networks. NFV has emerged as an innovative network paradigm that abstracts the network functions from hardware. NFV is closely related to other emerging technologies, such as Software Defined Networking (SDN) [4]. SDN is a networking technology that decouples the control plane from the underlying data plane and allows programmatic and centralized resource management of network functions. Combining SDN and NFV will enable to provide the complex network services in next-generation networks through centralized network management by SDN and specific abstraction and isolation mechanisms by NFV.

NFV enables telecommunications service providers (TSPs) to offer various network services by flexibly combining multiple virtual network functions (VNFs). These network services can be provided by combining multiple VNFs (e.g., virtual machines (VMs) and intrusion detection systems (IDSs)). The combination of VNFs is specified for each network service type. NFV can improve the efficiency of resource utilization by sharing physical resources between VNFs. To provide such services, a TSP should allocate virtual network

(VN) requests represented by virtual nodes as VNFs and virtual links connecting virtual nodes. A virtual node indicates the server resource requirements such as the required number of central processing units (CPUs) and amount of random access memory (RAM) and is often treated as a unit of a VM. A virtual link indicates the network resource requirements such as the required bandwidth and delay between virtual nodes.

1.1.2 Network resource management in edge computing

With the development of communication technologies, diverse applications have emerged in various domains, e.g., healthcare, smart cities, and manufacturing. These applications are generally offloaded and processed in cloud servers because of the limitation of the computation resources of end devices (EDs), e.g., personal computers, smartphones, Internet of Things (IoT) devices, and cars. This process is called cloud computing (CC). Such offloaded applications' tasks consist of the demand for computing and communication with various characteristics, e.g., traffic heavy, computing heavy, or latency sensitive. Since cloud servers are generally located far from EDs, offloading tasks to the cloud inevitably generates additional transmission latency. Therefore, CC degrades the performance of latency-sensitive tasks.

To address this issue, edge computing (EC) has been proposed to deploy computing resources at the edge servers close to EDs. Combining CC and EC provides multiple offloading choices, improving the efficiency of offloading. For example, offloading computing-heavy tasks to the cloud is effective because the cloud usually has sufficient computing resources. Similarly, offloading traffic-heavy and latency-sensitive tasks to the edge effectively shortens the transmission latency. Thus, CC and EC must be combined to improve the efficiency of various offloading tasks.

1.2 Challenges of network resource management

Network resource management, which involves mapping VN requirements and offloading tasks to the physical network, must be optimized to maximize re-

source utilization efficiency in a network with limited physical resources. It has been reported that the inefficient management of network policies accounts for 78% of data-center (DC) downtime [5,6]; therefore, resources must be optimally allocated to provide carrier-grade network services. The optimal management refers to the allocation of VNs and tasks that maximizes the objective function (e.g., resource utilization) while satisfying constraints (e.g., resource capacity). The performance of allocation algorithms is critical for future network management incorporating NFV, SDN, and EC technologies, as it determines the overall network's resource utilization efficiency. Though a significant amount of research has been conducted [7], no unified allocation problem has been formulated and solved that takes into account all conditions consisting of combination(s) of objective functions and constraints. In particular, many previous studies have proposed methods based on mathematical optimization, but the high computational cost of these methods makes it impractical to manage future networks.

1.3 Problem statements

This thesis studies four specific problems about network resource management using multi-agent deep reinforcement learning (MADRL). This thesis focused on reinforcement learning (RL) [8,9] as a solution to the resource management problem because it can quickly calculate a near-optimal resource allocation by learning the relationship between input network resource patterns and output resource allocation in advance. In addition, this thesis incorporates deep reinforcement learning (DRL) and/or multi-agent reinforcement learning (MARL). The definitions and formulations of RL algorithms and related works on RL-based network resource management are described in Chapter 2. Each problem corresponds to resource-integrated control in NFV, dynamic virtual network allocation, and task offloading for multi-cloud-edge networks, respectively.

1.3.1 Network-resource-integrated control in NFV

This thesis addresses the challenge of developing a network-resource-integrated control method in NFV, i.e., how to formulate and solve such a unified optimal

allocation problem. This problem becomes more difficult due to the increase in the number of control metrics (e.g., VNF placements and routes between the VNFs) and diversification of objective functions, where the control metric is defined by parameter(s) to characterize the state of a controlled network. In addition, the resource-integrated control method in NFV should be extendable, i.e., able to handle new control metrics being added or constraints of control metrics being changed. This requirement is crucial because a optimization problem needs to be solved quickly and easily even when a new network service starts or a new constraint needs to be taken into account. This thesis studies this problem in Chapter 3.

A simple way to achieve adequate extendability is to solve independently pre-specified optimization problems for each control metric. However, if we independently solve each optimization problem taking into account only the problem's constraints, we may not satisfy all the constraints. Hereafter, this thesis calls this problem control conflict (see Section 3.1 for details). To avoid control conflicts, resource-integrated control methods need to calculate the optimal allocation that satisfies all conditions determined by combination(s) of control metrics.

Previous studies on resource-integrated control methods can be categorized into two approaches. One is the combined approach [6, 10–16], which builds a specified algorithm that simultaneously solves the combined optimization problem. However, it is not extendable because we need to reconstruct the problem formula every time the combination of control metrics changes. The other is the coordinated approach [17–19], which involves using an extendable control architecture that coordinates multiple control algorithms pre-specified for individual control metrics. Though an extendable control architecture has been proposed, this architecture is only a concept and no specific implementation or formulation is described.

1.3.2 Dynamic virtual network allocation

The problem of finding an optimal VN allocation is known as the virtual network embedding (VNE) problem. Most existing approaches [7, 20, 21] only focus on static VN allocation, where the amount of VN demand for resources

is unchanged over time. When a VN is embedded at once in the physical network, the VN requests will hold fixed resources until the end of their lifetime in the static embedding process. However, since network traffic and computing demand have been changing dramatically due to the various types of network services, e.g., high-quality video delivery and operating system (OS) updates, the VN demands are dynamically changing and fluctuating. In the above situation, the static VN allocation leads to resources being inefficiently utilized and/or networks becoming congested, so the dynamic VN allocation for time-varying demand becomes more important. To more efficiently utilize resources, this work attempts to solve the dynamic VN allocation problem. Although the dynamic VN allocation problem has been studied for more than a decade, the following difficulty remains unresolved.

The difficulty in optimizing dynamic VN allocation is the need to simultaneously allocate VNs efficiently and immediately, even though efficiency and immediacy are in a trade-off relationship. Dynamic VN allocation needs to increase the computation time to increase the efficiency of VN allocation. The increase in computation time directly causes an increase in the control period. Alternatively, dynamic VN allocation requires a short control period to keep up with changes in demand. That is, increasing the efficiency leads to decreasing the immediacy. Similarly, since allocation performance decreases when the control period is limited, increasing immediacy leads to decreasing efficiency. Therefore, many methods have failed to simultaneously achieve efficiency and immediacy.

Several studies have addressed this problem [22–29]. Specifically, RL has been focused on as a solution [22–27]. RL solves the decision problem of what action an agent should take by observing the current state within a certain environment. An agent receives a reward from the environment depending on the selected action and learns a policy (i.e., strategy) that maximizes the received reward through a series of selected actions. RL is expected to be able to immediately output a close-to-optimal VN allocation even as the network resource demand drastically changes by learning the relationship between resource demand patterns and optimal VN allocation in advance.

There are two challenges in applying RL to dynamic VN allocation. One is related to safety, which is a common challenge in applying RL to real-world

applications. Satisfying constraints are important and never violating them is often required in real-world applications. However, there is no guarantee to satisfy constraints because the general RL agent learns the optimal action only by a reward. The agent receives a positive reward for preferable actions and a negative reward for unpreferable actions. This reward cannot fundamentally prohibit the action which does not satisfy the constraints. As an example of VN allocation, these constraints are equivalent to network congestion and server overload. This thesis studies this problem in Chapter 4.

The other is related to the number of candidate actions when solving the combinatorial optimization problem by RL, which is a problem-specific challenge for RL-based VN allocation. The RL approach relies on exploring all available actions sufficiently to compute a policy close to the optimal. Since the number of ways a VN can be embedded is combinatorial, the candidate actions of VN allocation exponentially increase as the number of nodes and the number of links increase. Therefore, the RL approach potentially requires a huge number of actions to derive an appropriate solution, which could lead to a prohibitively long convergence time for the learning process [30]. It was also reported that the performance of RL drastically worsens as the number of candidate actions increases [31]. Thus, the action space of the VNE problem needs to be shrunk. This thesis studies this problem in Chapter 5.

1.3.3 Task offloading for multi-cloud-edge networks

Several studies have addressed task-offloading problems for CC and EC [32–39]. Current methods based on mathematical optimization still incur high computational cost. For example, Wang *et al.* [32] accelerated the computation time by more than 100 times using their parallel optimization framework. However, due to the computational cost, they could only evaluate it in a small network with one cloud and a few base stations (BSs) (i.e., a few edge servers). To solve this problem, RL has been gaining attention as a solution [35–39]. RL can immediately output the preferred task offloading by learning the relationship between input network patterns and output task offloading in advance.

Although RL can solve the problem of computation time, two issues remain. One is that these methods [35–38] do not take into account CC or

target only the network with a single cloud. As mentioned above, CC and EC need to be combined to improve task-offloading efficiency. Moreover, a typical network has multiple clouds. The other issue is that these methods [35–39] do not take into account the bandwidth capacity and backbone-network topology. Many studies attempted to minimize task latency by shortening the path that offloaded tasks take. When a control system does not take into account bandwidth capacity, some links may become congested due to the concentration of tasks. Therefore, the system should obtain the task offloading that minimizes latency while satisfying all link-bandwidth constraints. Some of the above studies assumed a typology that is not practical as a backbone network, e.g., a full-mesh network. This thesis studies this problem in Chapter 6.

With RL-based task-offloading methods, multi-cloud and network topology are challenging to consider because route optimization requires many variables. Even a minimal network with five nodes requires hundreds of variables to optimize all routes between nodes. The performance of RL drastically worsens as the number of variables increases [31]. Therefore, computing offloading and route optimization are difficult to jointly solve by using RL-based methods. Since the joint problem is a non-deterministic polynomial time (NP)-hard mixed integer linear problem, it is also difficult to solve by mathematical optimization.

MARL is effective in solving more complex problems. It is a system of multiple agents interacting within a common environment. Each agent works with the other agent(s) to achieve a team reward. The learning cost of each agent can be reduced by assigning each agent to each task. However, when training decentralized and independent agents to optimize the team reward, each agent faces a non-stationary learning problem [40]. An example of this problem is that when each agent independently acts simultaneously, all tasks will be allocated on the light-load server, resulting in server overload. To avoid a non-stationary problem, Zhan *et al.* [35] developed an algorithm combining MARL and game theory. However, the actions of decentralized agents based on game theory fall into a sub-optimal solution of the Nash equilibrium. This thesis also studies this problem in Chapter 6.

Chapter 1: background and problem statements

Chapter 2: related works

Network resource management using multi-agent deep reinforcement learning

		Chapter 3	**Chapter 4**	**Chapter 5**	**Chapter 6**
Use Case		Virtual network allocation			Task offloading
Management Architecture		Network-resource-integrated control architecture proposed in **Chapter 3**			
Method	Reinforcement Learning	✓	✓	✓	✓
	with Multi-agent	✓	✓	✓	✓
	with Deep learning		✓	✓	✓
	with Safety		✓		
	with Cooperation			✓	✓

Chapter 7: conclusions and future works

Figure 1.1: Chapter overview of this thesis.

1.4 Overview and contributions of this thesis

Figure 1.1 shows the chapter overview of this thesis. Chapter 2 surveys the related works in literature.

Chapter 3 proposes an extendable resource-integrated control method based on the coordinated control architecture in NFV, which consists of multiple pre-specified control algorithms and a single coordination algorithm between the control algorithms. The key idea for extendability is modularization that divides a whole system into standardized functional elements and reduces the interdependence among the elements, which is a widely used technique for designing and managing huge complex systems. This work first prepares and solves each control algorithm for each control metric and then interactively im-

proves the results using the proposed coordination algorithm. This work also proposes an efficient coordination algorithm on the basis of RL. The learning makes it possible to learn the strategy for how to find better allocations efficiently from past exploration steps. The proposed method requires more iterations than the combined approach, but it achieves extendability through the coordinated control architecture.

In Chapters 4, 5, and 6, these works calculate the optimal computing allocation by RL and optimal routing by mathematical optimization. However, not all constraints may be satisfied when two methods are calculated individually. Therefore, this work proposes a combining method that uses the solution of mathematical optimization for RL reward, enabling all objectives and constraints of two methods to be considered. These methods are based by the architecture proposed in Chapter 3 that can coordinate multiple control algorithms prepared for each control metric.

Chapter 4 proposes a dynamic VN allocation method based on safe multi-agent deep reinforcement learning (Safe-MADRL). The main contribution of this work is the integration of the safety-considerations technique and multi-agent technique into a dynamic VN allocation method based on DRL. The proposed safety-considerations technique can reduce an agent's constraint violation. This work introduces two types of agents inspired by safety exploration [41]. One agent learns to improve the objective function (objective agent) and the other learns to satisfy all constraints (constraint agent). This work also decomposes the reward function dedicated to each agent and learn a action-value function for each component reward function, which is based on the hybrid reward architecture (HRA) [42]. Note that this work is also an RL-based approach, similar to previous methods [22–27]. Therefore, there is no guarantee that this work will satisfy the constraints. However, this work is safety-oriented, meaning it is based on RL but specifically designed to prioritize and achieve a higher level of safety. This work refers to "safety-oriented" simply as "safe." The proposed multi-agent technique can improve the performance and scalability for VN allocation. This work divides the VN demands into groups, and each agent is prepared for each group, which can reduce the number of candidate actions per agent. This work also restricted the agents that could act at each time to avoid conflicts among agents. An example of

such a conflict is when each agent independently acts at the same time, all VMs will be allocated on the smallest load server, resulting in server overload.

Chapter 5 proposes a dynamic VN allocation algorithm based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). The key idea is to use a multi-agent technique for an RL-based dynamic VN allocation method. This work prepares each agent for each VN allocation control, which can reduce the action space. However, when training decentralized independent agents to optimize for the team reward, each agent is faced with a non-stationary learning problem, i.e., the dynamics of its environment change as other agents change their behaviors through learning [40]. An example of such a non-stationary problem is that when each agent independently acts at the same time, all VMs will be allocated on the smallest load server, resulting in server overload. Therefore, this work introduces a cooperative element in which several agents jointly optimize a single reward through centralized training and decentralized execution, which can improve the efficiency of VN allocation. The proposed method directly treats all candidate actions at each step without restricting agent actions or modeling and compressing handcrafted features for states and actions. This cooperation can avoid conflicts of control between agents and improve the performance of VN allocation. Moreover, it can take the hassle out of problem-specific feature design.

Chapter 6 formulates an optimal task-offloading problem and proposes a task-offloading method for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. This work handles the task offloading of edge-to-edge and edge-to-cloud. Each task arriving at the nearest edge is either processed at the nearest edge or offloaded to the neighboring edge or cloud. This work defines optimal offloading as a solution that maximizes server- and link-resource efficiency and minimizes task latency while satisfying the constraints of server and link capacity and task latency. The decision variables are the computing-resource allocation of tasks and routing between the ED and allocated server. The key solution to this problem is that mathematical optimization can quickly solve the route-optimization problem. Therefore, this work develops a method for combining RL and mathematical optimization. This work assigns an agent at each edge that has learned the optimal task offloading. This work introduces a cooperative multi-agent tech-

nique proposed in Chapter 5 in which several agents jointly optimize a single reward in a centralized training and decentralized execution manner. This can prevent the non-stationary problem and improve task-offloading efficiency. The proposed method combines Coop-MADRL and mathematical optimization to take into account network topology and bandwidth constraints in the task-offloading problem. This work uses the solution of mathematical optimization in the learning process of Coop-MADRL. The proposed method calculates the optimal computing offloading by RL and the optimal routing by mathematical optimization. Therefore, the proposed method can handle numerous routing variables without approximating or reducing them.

Finally, Chapter 7 concludes this thesis and discusses the future works to extend this work.

Chapter 2

Related works

This chapter is structured as follows. Section 2.1 briefly reviews reinforcement learning (RL). Section 2.2 describes the related works in network-resource-integrated control. Section 2.3 describes the related works in dynamic virtual network (VN) allocation. Section 2.4 describes the related works in task-offloading for cloud computing (CC) and edge computing (EC).

2.1 Reinforcement learning

This section briefly reviews single-agent RL, MARL, and safe RL.

2.1.1 Single-agent reinforcement learning

Single-agent RL considers a sequential decision-making problem in which an agent interacts with an environment. The agent observes state $s \in \mathcal{S}$ in which $\mathcal{S}$ is the state space, takes action $a \in \mathcal{A}$ where $\mathcal{A}$ is the action space, and executes it in the environment to receive reward r and transfer to the new state $s' \in \mathcal{S}$. The goal of the agent is to determine a policy that maximizes the long-term reward. The policy is a map $\pi : \mathcal{S} \to P(\mathcal{A})$, where P is the transition probability among the states. Q-learning [8], a widely used RL algorithm, learns the relationship of $\langle s, a, r, s' \rangle$ to maximize the action value $Q(s, a)$, which is defined as the expectation of the sum of rewards obtained in the future when action a is selected in state s. The agent receives a reward r

and updates the Q-function in accordance with the following equation.

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') \right], \tag{2.1}$$

where α is a learning rate and γ is a discount factor.

Deep reinforcement learning (DRL) [9,31] dramatically improves the generalization and scalability of traditional RL algorithms and can handle continuous and high-dimensional state space by approximating the $Q(s,a)$ with a deep neural network (DNN). Among the numerous studies on DRL, one prominent study is the Deep Q-network (DQN) [9], which has demonstrated human-level control and achieved outstanding performance in various classic games. Another notable study is the Deep Deterministic Policy Gradient (DDPG) [43], which adopts the successful principles of Deep Q-Learning and applies them to the continuous action domain. DDPG has demonstrated robust performance in more than 20 simulated physics tasks using the same learning algorithm, network architecture, and hyperparameters.

This thesis adopts the DQN as the base for the DRL algorithm because it is well-suited for handling discrete actions and has demonstrated stability and high performance across various scenarios. The optimal resource management problem involves handling discrete actions as it requires selecting the best server from a set of available servers. Note that, similar to previous studies [9, 43, 44], this thesis defines DRL as the RL with multiple hidden layers of the DNN and does not restrict the number of DNN layers.

DQN uses a *replay memory* to store the transition tuple $\langle s, a, r, s' \rangle$. DNN parameters θ are learned by sampling batches b of transitions from the replay memory and minimizing the squared error:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\mathrm{DQN}} - Q(s,a;\theta) \right)^2 \right], \tag{2.2}$$

$$y^{\mathrm{DQN}} = r + \gamma \max_{a' \in \mathcal{A}} Q(s',a';\theta^-), \tag{2.3}$$

where θ^- are the parameters of a *target network* that are periodically copied from θ and kept constant for several iterations. It was reported that the performance of RL algorithms drastically worsens as the number of candidate actions increases due to the decrease in the sampling efficiency of $\langle s, a, r, s' \rangle$

and increase in the error of the DNN [31]. Therefore, stable DRL becomes difficult as the number of actions increases.

DRL performs better when historical information is used for learning. DQN defines the last four frames as the current state in the classic game environment to consider historical information. This approach was successful enough for games where reflexes are critical, but it became clear that DQN needed more than four frames for some games. To address these shortcomings, Deep Recurrent Q-Network (DRQN) [45] adds recurrency to DQN by replacing the first post-convolutional fully connected layer with a recurrent long short-term memory (LSTM). DRQN successfully integrates historical information and improves the performance of DQN in some games, even though it only sees a single frame at each time step. In addition, DRQN with the recurrent network can better adapt during an evaluation when the quality of the observations changes.

2.1.2 Multi-agent reinforcement learning

MARL is a system of multiple agents interacting within a common environment. Each agent decides each time-step and works together with the other agent(s) to achieve a given goal. It is used for learning a complex environment by dividing a single task into multiple sub-tasks. The learning cost of each agent can be reduced by assigning each agent to each task. The recent review of MARL is described in detail [46–48]. Due to the complexities of the environments and the combinatorial nature of the problem, most MARL problems are categorized as NP-hard problems [49].

A cooperative multi-agent environment can be described as a decentralized partially observable Markov decision process (Dec-POMDP) [50] consisting of $\langle K, \mathcal{S}, \mathcal{A}, R, P, \mathcal{O}, \gamma \rangle$, in which K is the number of agents, $\mathcal{S}$ is state space, $\mathcal{A} = \{\mathcal{A}^1, \ldots, \mathcal{A}^K\}$ is the set of actions for all agents, P is the transition probability among the states, R is the reward function, and $\mathcal{O} = \{\mathcal{O}^1, \ldots, \mathcal{O}^K\}$ is the set of observations for all agents. In a cooperative multi-agent problem in which the environment can be fully observed, the i^{th} agent at time-step t observes the global state s_t, takes action a_t^i ($\boldsymbol{a}_t = \{a_t^i\}$), and receives reward r_t. If the agents cannot observe the global state, each agent only accesses its own

local observation o_t^i. The state s_t needs to contain all information required to uniquely represent the current environment's status. Whereas the observation o_t^i is a part of state s_t, and it needs to contain the information required to uniquely represent i^{th} agent's current status.

Each agent has an observation-action history $h^i \in \boldsymbol{h}$, where the history indicates a series of past observations and $\boldsymbol{h}$ is the observation-action history of all agents. The joint policy is a map $\pi : \boldsymbol{h} \to P(\mathcal{A})$, and the joint policy π has a joint action-value function:

$$Q^{\pi}(\boldsymbol{h}_t, \boldsymbol{a}_t) = \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^j r_{t+j} \mid \boldsymbol{h}_t, \boldsymbol{a}_t\right]. \tag{2.4}$$

In the following section, this work describes several significant MADRL algorithms based on the DQN that can handle discrete actions. However, this work does not describe the major MADRL algorithms for continuous actions, e.g., Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [51].

Independent Q-learning

The most naive approach to solve the MARL problem is to treat each agent independently. This idea is formalized in an independent Q-learning (IQL) algorithm [52, 53], which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. Each agent runs Q-learning [8] or DQN [9]. IQL is scalable from the viewpoint of implementation while increasing the number of agents, and each agent only needs its local history of observations during the training. In this theise, IQL is trained to minimize the loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \sum_{k=1}^{K} \left[\left(y_i^k - Q_k(h^k, a^k; \theta^k)\right)^2\right], \tag{2.5}$$

where b is the batch size of transitions sampled from the replay memory, Q_k is the k^{th} agent's Q-value function, and y^k is the k^{th} agent's y^{DQN} as in Eq. (2.3).

Value decomposition networks

Value decomposition networks (VDNs) [54] aim to learn a joint action-value function $Q_{tot}(\boldsymbol{h}, \boldsymbol{a})$. It is assumed that the joint action-value function Q_{tot} can

be additively decomposed into K Q-value functions for K agents, in which each Q-value function Q_i only relies on the local state-action history:

$$Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) = \sum_{i=1}^{K} Q_i(h^i, a^i; \theta^i). \tag{2.6}$$

Therefore, each agent observes its local state, obtains the Q-values for its action, and selects an action, and then the sum of Q-values for the selected action of all agents provides the total Q-value of the problem. By using the shared reward and the total Q-value, the loss is calculated and then the gradients are back-propagated into the networks of all agents. Because each agent's DNN is updated on the basis of the total Q-value, each agent learns the best behavior for all agents, i.e., they learn cooperative behavior. The loss function for VDN is as follows:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\boldsymbol{h}, \boldsymbol{a}; \theta) \right)^2 \right], \tag{2.7}$$

where b is the batch size, $y^{tot} = r + \gamma \max_{\boldsymbol{a}'} Q_{tot}(\boldsymbol{h}', \boldsymbol{a}'; \theta^-)$ and θ^- are the parameters of a target network as in DQN.

QMIX

QMIX [55] extends VDN to address a broader class of environments. To represent a more complex factorization, a mixing network with trainable parameters is introduced to compute the total Q-value on the basis of each agent's state-action value function. As mentioned above, VDN adds restrictions to have the additivity of the Q-value and further shares the action-value function during the training. QMIX also shares the action-value function during the training. Besides, QMIX adds the below constraint to the problem:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i, \tag{2.8}$$

which enforces positive weights on the mixer network, and as a result, QMIX can guarantee monotonic improvement. QMIX is trained to minimize the DQN loss, and the gradient is back-propagated to the individual Q-values, similarly to VDN.

2.1.3 Safe reinforcement learning

Dalal at el. [41] proposed an RL algorithm for a physical system where critical constraints must never be violated, such as a data-center cooling unit. Their approach relies on a one-time initial pre-training of a model that predicts the change in the safety signal over a single time-step. The model is a first-order approximation with respect to the action, where its coefficients are the outputs of a DNN. This model is used in a safety layer composed directly on the agent's policy to correct the action if needed, i.e., after every policy query, it solves an optimization problem for finding the minimal change to the action such that the safety constraints are met.

2.1.4 Hybrid reward architecture

In domains where the optimal value function cannot easily be reduced to a low-dimensional representation, learning can be very slow and unstable. Seijen *et al.* [42] tackled such challenging domains by using HRA. They decomposed a reward function and learned a separate value function for each component reward function. The reward function is decomposed into n reward functions with weighting parameters w_k, and action selection is based on the sum of each Q-value function Q_k of k^{th} agent:

$$Q_{\text{HRA}}(s,a) := \sum_{k=1}^{n} w_k Q_k(s,a). \tag{2.9}$$

2.2 Network-resource-integrated control

Previous studies on resource-integrated control methods can be categorized into two approaches: combined and coordinated.

2.2.1 Combined approach

The combined approach [6, 10–16] builds a specified algorithm that simultaneously solves the combined optimization problem.

Jiang *et al.* [10] studied a combined optimization problem of VM placement and routing to minimize traffic costs in an intra-DC. They also proposed an ef-

ficient online algorithm in a dynamic environment under changing traffic loads by leveraging and expanding the technique of Markov approximation. However, since the algorithm approximates the combined optimization problem by utilizing the specific problem structure, it is difficult to extend to another use case.

Yoshida *et al.* [11] designed a plug-in architecture to satisfy various requirements related to NFV resources. They also proposed a modified multi-objective genetic algorithm (MOGA) to obtain approximate solutions in reasonable computation time. However, each plug-in should be pre-formulated as a format of objective functions and/or constraints of MOGA. Moreover, they evaluated only one use case and did not mention extendability for adding and/or changing plug-ins.

Jin *et al.* [12] proposed optimization method to minimize the cost of caching, transcoding, and routing functions for cost-efficient video distribution over the future Internet. They developed two algorithms for maximizing total cache hits and minimizing the networking cost. They improve the solution by alternately calculating those two algorithms. However, this method cannot be applied to general use cases due to the control conflict between objective functions (a detailed example is given in Section 3.1).

Cui *et al.* [6] formulated the Policy-VM Consolidation problem, which can jointly optimize the VM placement as the origin/destination node and VNF placement as the middle node, and the route between VMs via VNF. They also proposed an efficient and synergistic scheme to jointly consolidate VNFs and VM. However, this scheme is heuristic and specified for only one use case, so it is difficult to extend to other use cases.

Herrera *et al.* [7] survey the research challenges of solving the resource allocation problem in NFV-based network architectures. They classify the NFV resource allocation problems into three stages: VNFs chain composing, VNF forwarding graph embedding, and VNFs scheduling. They mention that these three stages of the NFV resource allocation problem are related to each other, and a way to coordinate the three stages is a major challenge of the NFV resource allocation problem. The aim is to optimize the use of resources to improve the performance of the network.

To coordinate NFV resource allocation problems, various studies have been

conducted [13–16]. In particular, Li *et al.* [16] formulated a typical three-stage coordinated NFV resource allocation model as a mixed integer programming (MIP) and proposed a heuristic solution called merge-split viterbi (MSV). However, MSV is a specified algorithm that simultaneously solves the combined optimization problem, so it is difficult to extend to other use cases. Other algorithms [13–15] are similar to the case of MSV.

2.2.2 Coordinated approach

The coordinated approach [17–19] involves using an extendable control architecture that coordinates multiple control algorithms pre-specified for individual control metrics.

Tsagkaris *et al.* [17, 18] and Stamou *et al.* [19] proposed a hierarchical network control framework for unifying all control when a network contains multiple control metrics. In particular, their proposed architectures [18, 19] include a single Autonomic Network Management (ANM) Core and multiple Autonomic Control Loops (ACLs). Each ACL is a control module specialized for one control metric. The ANM Core integrates control of all ACLs and determines whether each ACL appropriately controls each control metric. However, this extendable control architecture is only a concept, and no specific implementation or formulation is described.

2.3 Dynamic virtual network allocation

There have been several studies on dynamic VN allocation (RL-based [22–27] and heuristic [28, 29]).

2.3.1 Reinforcement learning-based allocation

Mijumbi *et al.* [22–24] proposed a MARL-based dynamic bandwidth control method under the decentralized resource management system, which prepares the agents for each physical node and physical link. They [22] applied a Q-learning based RL agent. They [23] also used an artificial network to make resource reallocation decisions and train the network with a Q-table. They [24]

also proposed an RL-based neuro-fuzzy algorithm. However, it controls only the buffer size of virtual nodes and the bandwidth of virtual links and does not reallocate virtual nodes. Therefore, it cannot cope with demand changes for server resources. Moreover, to prevent the number of actions from exponentially increasing, all the demands passing through each node and link are controlled by uniform parameters. Therefore, the bandwidth for all users is limited regardless of the demand of each user.

With the breakthrough of DRL in human-level control applications [9], the DRL-based VNE algorithm has been increasingly studied. Studies have mentioned the problem of the RL-based approach: the candidate actions of VN allocation exponentially increase as the number of nodes and the number of links increase [25–27]. Dolati *et al.* [25] attempted to shrink the action space of the VNE problem to provide sufficient flexibility for exploring different VN mappings while retaining the efficiency of the learning process. They adopted a convolutional neural network for a DRL approach in solving VNE problems. Dolati *et al.* assumed that networks have grid-like typologies to make the image representation easier to obtain, but this is not true in many other situations. Yan *et al.* [26] decomposed a VNE process into a sequence of virtual node embedding to shrink the action space, and the learning agent only focuses on one virtual node of the current VN request at every single step. The work in Chapter 4 proposed a MADRL-based dynamic VN allocation method [27]. This work divided the VN demands into groups, and each agent is prepared for each group, which can shrink the action space per agent. This work also restricted the agents that could act at each time to avoid conflicts among agents. However, the shrinking action space and the restriction on the agent's action decrease the performance of VNE. Moreover, this restriction increases the number of steps required for VN reallocation and may delay the response to dynamic demand changes. Conversely, the work in Chapter 5 adds a cooperative element to the previous method, which can avoid conflicts of control between agents and improve the performance of VN allocation.

In conclusion, the shortcoming of existing methods and the strengths of the proposed method is summarized. As mentioned above, the problem of RL-based approaches is that the number of candidate actions for VN allocation increases exponentially as the number of nodes and links increases. Exist-

ing methods introduce the multi-agent technique to prevent the exponential increase of actions. However, existing methods restrict the agent's actions at each step or compress the action space using handcrafted features, which degrades the performance of VN allocation. On the other hand, the work in Chapter 5 additively introduces a cooperative technique for a multi-agent DRL-based dynamic VN allocation method. This technique can improve the performance of VN allocation without restricting agent actions or compressing actions.

2.3.2 Heuristic-based allocation

Zheng *et al.* [28] proposed a dynamic VNE algorithm based on a radial basis function neural network to learn and predict the dynamic changes of resources, and users dynamically adjust and allocate resources by the predicted results. Although these proactive control methods based on predicting the dynamic changes are effective for near-stationary resource demand, they sometimes fail since drastic demand changes cannot be predicted. On the other hand, the proposed approach uses the almost real-time demand for control by drastically reducing the computation time and control interval, not minimizing the prediction error.

Dehury and Sahoo [29] proposed a dynamic VN allocation method that combines online and offline allocation algorithms. The VN demand is accepted and allocated immediately by the online algorithm, then VN demand is periodically reallocated by the offline algorithm. By combining the two algorithms, they simultaneously achieved immediate demand acceptance and optimal demand allocation. However, they assumed only adding and deleting virtual links and not the change in virtual link bandwidths. Therefore, their method cannot be applied for time-varying traffic demand.

Table 2.1: Comparison of related studies.

Studies	Network Properties				Method Properties		Evaluation Properties	
	# of Edges	# of Clouds	Topology	Access Network	Methodology	Cooperation	# of Task Types	Comprehensiveness
[32]	Several	Single	-	Wireless	ADMM	✓	Single	Limited
[33]	Dozens	Single	-	Wireless	SA	-	Single	Limited
[34]	Several	Single	-	Wireless	Pipeline Strategy	-	Single	Limited
[35]	Several	-	-	Wireless	MADRL	-	Single	Limited
[36]	Dozens	-	-	Wireless	MADRL	✓	Single	Limited
[37]	Several	Single	-	Wireless	MADRL	✓	Single	Limited
[38]	Several	Single	-	Wireless	MADRL	✓	Single	Limited
[39]	Several	Multi	-	Wireless	MADRL	-	Single	Limited
Ours	Dozens	Multi	✓	-	MADRL	✓	Multi	✓

2.4 Task-offloading for cloud computing and edge computing

Several studies have addressed task-offloading problems for CC and EC [32–39]. In general, CC has sufficient computing resources but inevitably increases network latency. On the other hand, EC can reduce network latency at the expense of having sufficient computing resources. These studies aim to optimize the task-offloading when considering the characteristics of heterogeneous computing resources. They sometimes refer to fog computing instead of CC or EC, or assume multiple layers in the edge network. When an edge fog or a two-tier edge network considers the heterogeneous resource characteristics, the work in Chapter 6 can see them as the same studies.

Table 2.1 summarizes the characteristics of these studies and ours. "Cooperation" in method properties means whether multiple control algorithms are coordinated when a method contains multiple algorithms. It is not the case when a single algorithm determines all task offloads or when multiple algorithms independently determine them. "Comprehensiveness" in evaluation properties means whether the methods evaluate the performance of the algorithm under practical conditions, e.g., a sufficient number of edges and clouds, and under various metrics, e.g., computation time and scalability.

Wang *et al.* [32] considered a cooperative three-tier computing network by leveraging vertical cooperation among devices, edge nodes, and cloud servers and horizontal cooperation between edge nodes. They also presented a parallel optimization framework by using the alternating direction method of multipliers (ADMM) method. However, they did not impose network bandwidth constraints and did not assume multi-cloud networks. They evaluated minimal conditions with four edges and 40 devices. Yuan *et al.* [33] designed a profit-maximizing collaborative computation offloading and resource-allocation algorithm to maximize system profit and guarantee task-response-time constraints. They also developed a migratory-bird optimization method that is based on simulated annealing (SA) to obtain a close-to-optimal solution. However, they did not assume multi-cloud networks and backbone-network topology. They evaluated their method under very light conditions, such as one task every 20 seconds. Kai *et al.* [34] developed a collaborative computing framework to pro-

cess mobile devices' tasks at terminals, edge nodes, and cloud centers. They presented a pipeline-based offloading scheme, in which both mobile devices and edge nodes can offload computation-intensive tasks to an edge node and cloud center in accordance with their computation and communication capacities, respectively. However, they did not assume multi-cloud networks and backbone network topology and evaluated minimal conditions with 40 devices.

MADRL has gained attention as a solution to the problem of computation time [35–39]. Zhan *et al.* [35] designed a decentralized algorithm for computation offloading so that users can independently choose their offloading decisions. They developed their algorithm by combining MADRL and game theory. However, the actions of decentralized agents based on game theory fall into a sub-optimal solution of the Nash equilibrium. They only considered EC and did not consider CC. Nguyen *et al.* [36] presented a new collaborative offloading framework that is based on MADRL in heterogeneous edge networks where each ED acts as an intelligent agent to make offloading decisions collaboratively and achieve optimal system utility. However, they only considered EC and did not consider CC. Hou *et al.* [37] introduced a hierarchical task-offloading and resource-allocation method that is based on MADRL for the Cybertwin-based network. It can promote the flexible collaboration of EDs, EC servers, or CC servers to improve system processing efficiency and security. Although they described the formulation in detail, their work remains only a concept, and they evaluated minimal conditions with a single cloud, three edges, and 100–300 devices. Ding *et al.* [38] considered cooperative task offloading, where all edge servers cooperate to achieve good performance for the entire edge-CC system, such as low latency and energy costs. They addressed MADRL-based task offloading that takes into account cooperation among agents. However, they evaluated minimal conditions with a single cloud and five edges. They evaluated only one snapshot and did not evaluate statistical performance. They also did not conduct any practical evaluation, e.g., computation time or scalability. Zhang *et al.* [39] considered a three-layer distributed multi-access edge computing network where there are multiple clouds, EC servers, and EDs. They developed a distributed scheme that is based on MADRL. Each cloud jointly determines the offloading task and resource-allocation strategy on the basis of its inference of other cloud decisions. However, they evaluated minimal

conditions with 1–3 clouds, 1–3 edges, and 3–9 devices.

In summary, previous studies did not consider CC or targeted only the network with a single cloud. Zhang *et al.* [39] assumed multi-cloud networks, but their evaluation was under the limited condition of three edge nodes, and effectiveness of their method under practical conditions remains unclear. Previous studies did not consider the link capacity and topology of the backbone network. They assumed that the backbone-network link between clouds and edges was as if it were a single link. To the best of our knowledge, the work in Chapter 6 is the first to focus on optimal task offloading for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. Several studies have addressed MADRL-based task offloading by considering cooperation among agents similar to ours but do not meet the above network requirements. Furthermore, this work introduces a generalized task model representing various task types. Previous studies evaluated the performances of their methods under conditions where they generated only one type of task uniformly. This work evaluated the effectiveness of the proposed method in terms of performance, network-topology dependency, computation time, scalability regarding the number of tasks, and generalization performance for unknown task patterns. Such a comprehensive evaluation had not been conducted.

Chapter 3

Extendable resource-integrated control using reinforcement learning

Network functions virtualization (NFV) enables telecommunications service providers to realize various network services by flexibly combining multiple virtual network functions (VNFs). To provide such services, an NFV control method should optimally allocate such VNFs into physical networks and servers by taking account of the combination(s) of objective functions and constraints for each metric defined for each VNF type, e.g., VNF placements and routes between the VNFs. The NFV control method should also be extendable for adding new metrics or changing the combination of metrics. One approach for NFV control to optimize allocations is to construct an algorithm that simultaneously solves the combined optimization problem. However, this approach is not extendable because the problem needs to be reformulated every time a new metric is added or a combination of metrics is changed. Another approach involves using an extendable network-control architecture that coordinates multiple control algorithms specified for individual metrics. However, to the best of our knowledge, no method has been developed that can optimize allocations through this kind of coordination. This work proposes an extendable network-resource-integrated control method in NFV by coordinating multiple control algorithms. This work also proposes an efficient

coordination algorithm based on reinforcement learning. Finally, this work evaluates the effectiveness of the proposed method through simulations. A part of this work in this chapter was presented in [56, 57].

This chapter is structured as follows. Section 3.1 describes the motivation of this work in detail. Section 3.2 describes the proposed extendable resource-integrated control method in NFV and an efficient algorithm for the proposed method using RL. Section 3.3 describes the use cases for the proposed method, the modeling and formulation of the proposed method, and its extendable implementation. Section 3.4 evaluates the performance, and Section 3.5 concludes the chapter.

3.1 Challenges and motivation

In this section, this work describes the challenges and motivation for extendable resource-integrated control methods with a concrete use case. This work first considers the use case as an example in which this work provides a secure-cloud-computing service consisting of routes between VMs via an IDS. In this case, the control metrics are routes, VM placements, and IDS placements, and each control algorithm is pre-formulated (detailed formulation is given in Section 3.3.3).

Independently solving each optimization problem leads to the following control conflicts.

(1) Conflict between constraints - Capacity overload: Since each control algorithm takes into account only its constraints, all constraints might not be satisfied at the same time. For example, when each problem for each control metric is independently solved at the same time, VMs and IDSs will be allocated on the same server, resulting in server overload.

(2) Conflict between objective functions - Oscillatory solution: If each algorithm with a different objective function is conducted independently, the network may become unstable. For example, if the IDS-allocation algorithm to balance server loads and the VM-allocation algorithm to minimize electric power consumption (i.e., the number of powered-on servers) are used independently (e.g., the latter is done after the former repeatedly), most assignment results are repeatedly changed.

The above conflicts can be avoided by sequentially solving each optimization problem for residual resources of each allocated result. However, the obtained results after conducting all the algorithms are not guaranteed to become optimal. This is because, since the result of the previous algorithm is fixed, an inefficient solution may be inevitable. For example, when the results of the physical distance between allocated VMs are long, inefficient routing is inevitable.

The motivation for this work is to avoid the conflicts and inefficiency described above. In addition, this work addresses the challenge of extendability for changing control metrics that are considered essential in resource-integrated control. The goal is to construct an extendable resource-integrated control method, i.e., enabling NFV control metrics to be changed and added without changing each control-algorithm formulation. Though the proposed method cannot solve all the challenges of extendability or cover all use cases, to the best of our knowledge, this is the first work to tackle this problem, and this work expects that more complicated use cases can be solved by enhancing the proposed method.

3.2 Proposed method

This work has developed an extendable resource-integrated control method by coordinating multiple control algorithms. This work has also developed an efficient coordination algorithm by using RL to find better solutions with fewer coordinating iterations than the case without RL.

In this section, this work first describes the overview and procedure of the proposed method and then gives a formulation of the proposed coordination algorithm.

3.2.1 Overview of proposed method

The proposed method executes hierarchical control consisting of multiple **control engines** and a single **coordination engine** (Fig. 3.1). A control engine has an algorithm to calculate a solution for each control metric and calculate the evaluation value of the solution quantitatively. The evaluation value of a

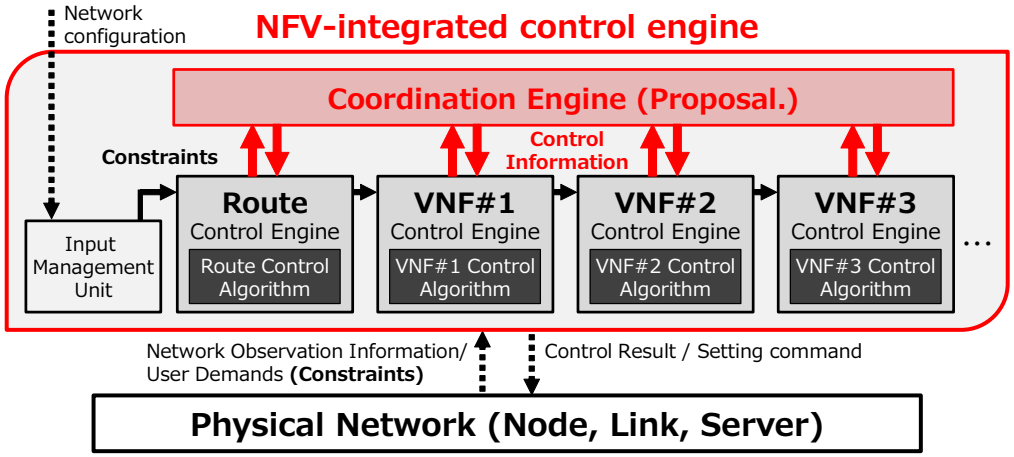


Figure 3.1: Overview of proposed resource-integrated control method. (©2020 IEICE.)

solution is defined as the objective-function value if the constraints of a control algorithm are satisfied; otherwise, it returns a negative value as a penalty. The objective-function value allows only positive values. Using this negative value, this work can determine whether the constraints are satisfied. The coordination engine explores a solution by changing a part of the solution to improve the **comprehensive evaluation value (CEV)**, which is defined as a unique value determined by all evaluation values of solutions calculated by the individual control engines, e.g., the weighted average of each evaluation value of the solution. The weight of each evaluation value is determined from the importance of each objective function.

This work describes the procedure of the proposed method. Each control engine first calculates initial solutions independently, and then the proposed coordination engine recursively explores the solutions to improve the CEV. In the exploration procedure, the coordination engine first changes a part of a solution on the basis of the current CEV, and then the changed solution is sent to each control engine. Next, each control engine calculates the evaluation value on the basis of the changed solution. At this time, some control engines calculate the part of the next solution together as necessary. For example, a route control engine needs to calculate the next route on the basis of the changed VNF placements. The coordination engine calculates the next CEV

on the basis of the evaluation value and then returns to the beginning of the procedure. When the exploration is terminated by repeating the above procedure a certain number of times, this work regards the highest CEV solution among the past iterations as the final solution. The proposed method can be extended because this work improves each solution on the basis of only the CEV, independently of the control metric type or number of control engines.

3.2.2 Overview of coordination algorithm

RL solves the decision problem of what **action** an "agent" should take by observing the current **state** within a certain "environment." An agent receives a **reward** from the environment depending on the selected action and then learns a strategy for how to maximize the received reward through a series of selected actions. In a resource-integrated control environment, an agent's strategy indicates an efficient solution exploration to improve the CEV, and the agent observes the current solution and CEV. This work bases the proposed algorithm on RL because general-purpose learning is possible by just defining states, actions, and rewards and applying them to general control engines without a specific algorithm.

Specifically, this work uses hierarchical MARL [58], consisting of a single **instruction agent** and multiple **control agents** (Fig. 3.2). The instruction agent learns a selection of the control agent, and the control agent learns an efficient solution exploration for the corresponding control engine. The purpose of hierarchical MARL is to make the proposed method extendable by preparing a specific control agent for each control engine. Since all agents' learning algorithms are common, the learning algorithm is not affected by any change of any control algorithm.

This work introduces the input/output (I/O)-conversion unit, which converts the I/O format of each control engine. Because the changed solution of a control engine may affect evaluation values of other control engines, this work needs to share the changed solution by converting the I/O format. For example, the VNF placements affect the routes between the VNFs, so this unit needs to convert the VNF placements (i.e., an output of a VNF control engine) into traffic demands between servers on which the VNFs are allocated (i.e., input

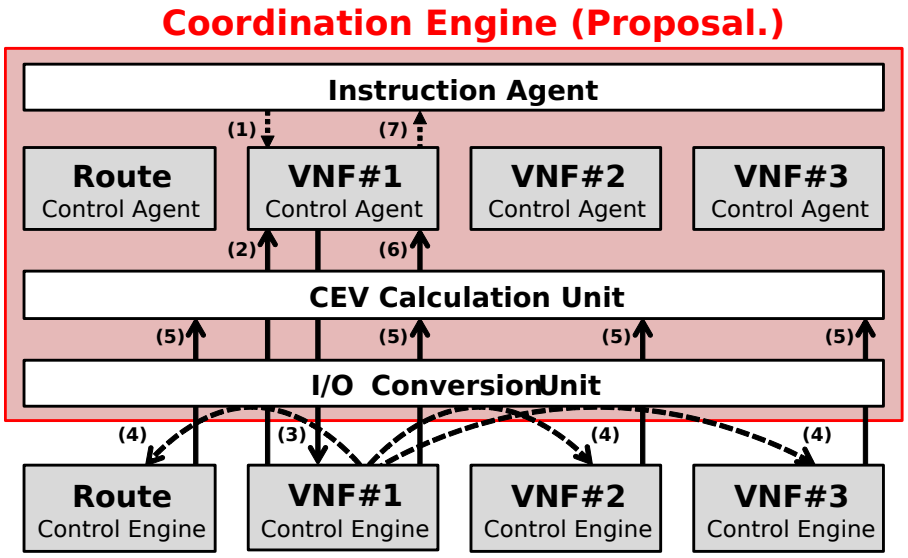


Figure 3.2: Overview of coordination engine based on reinforcement learning. An example when instruction agent selects VNF#1 control agent. (©2020 IEICE.)

of a route control engine). This work assumes that the cost of implementing the I/O conversion unit is lower than the cost of rebuilding each algorithm formulation. An example of the I/O-conversion is described in Section 3.3.3.

This work describes the procedure of the proposed coordination algorithm based on hierarchical MARL. As shown in Fig. 3.2, the instruction agent first selects a control agent on the basis of the RL (1), and then the selected control agent starts exploring solutions and learning a strategy of exploration (2)–(6). In the exploration step, the selected control agent observes the current solution (2), then changes a part of the solution on the basis of the RL, and sends the changed solution (3). After the changed solution is shared through the I/O-conversion unit (4), each control engine calculates its evaluation value. Next, the CEV is calculated from all evaluation values (5), and then the control agent receives the CEV as a reward (6). Then, the instruction agent learns the strategy of selecting a control agent on the basis of the maximum CEV in the exploration (7) and selects the next agent on the basis of the strategy. After repeating the procedure, the best solution is output as the final solution. Finally, the procedure returns to the beginning (1).

Table 3.1: Symbol descriptions for coordination algorithm.

Symbols	**Definitions**
ia	Instruction agent
$\boldsymbol{G} := \{g\}$	Control agent set
$\boldsymbol{E} := \{e\}$	Control engine set
$t \in T$	Exploration step (T: Total exploration steps)
t^g	Number of iterations of control agent ($g \in \boldsymbol{G}$)
T^g	Total exploration steps of control agent ($g \in \boldsymbol{G}$)
s_t^{agent}	State of each agent at step t (agent $\in$ ia $\cup \boldsymbol{G}$)
a_t^{agent}	Action of each agent at step t (agent $\in$ ia $\cup \boldsymbol{G}$)
r_t^{agent}	Reward of each agent at step t (agent $\in$ ia $\cup \boldsymbol{G}$)
$Q\left(s_t^{\text{agent}}, a_t^{\text{agent}}\right)$	Policy value for state s_t^{agent} and action a_t^{agent}
α, γ	Hyper-parameter (Default: $\alpha = 0.2$ and $\gamma = 0.9$)
$\boldsymbol{A}^e$	Solution of control engine e
$\boldsymbol{D_t} := \{d_{ijt}\}$	Traffic demands from node i to node j at step t
$\boldsymbol{V_t} := \{v_t^e\}$	Evaluation values of control engine e at step t
$\boldsymbol{\theta} := \{\theta^e\}$	Coefficients of control engine e

3.2.3 Formulation of coordination algorithm

This work describes the formulation of the proposed coordination algorithm. Table 3.1 summarizes the definitions of the variables of the proposed coordination algorithm. For agent learning algorithms, this work uses Q-learning [8], which learns the relationship of a state, action, and reward to maximize the policy value. Policy value $Q(s_t, a_t)$ is defined as the expectation of the sum of rewards obtained in the future when action a_t is selected in state s_t.

Instruction-agent algorithm

The instruction agent learns how to select control agents. A state is defined as the selected control agent, action as the selection of the next control agent, and reward as the maximum CEV obtained during this control-agent selection.

The instruction-agent algorithm is shown in Algorithm 1. Lines 1–2 show the initialization of $Q\left(s^{\text{ia}}, a^{\text{ia}}\right)$, exploration step t, and initial state s_0^{ia}. The

Algorithm 1 Instruction-agent Learning.

1: **initialize:** $Q\left(s^{\text{ia}}, a^{\text{ia}}\right) \leftarrow 0$, for all s^{ia} and a^{ia}
2: **initialize:** $t \leftarrow 0, s_0^{\text{ia}} \leftarrow$ random choice from $\boldsymbol{G}$
3: **while** $t < T$ **do**
4: $\quad a_t^{\text{ia}} \leftarrow \epsilon$ greedy $\left(s_t^{\text{ia}}\right)$
5: $\quad s_{t+1}^{\text{ia}} \leftarrow$ action $\left(a_t^{\text{ia}}\right)$
6: $\quad r_{t+1}^{\text{ia}}, t^g \leftarrow$ agent learning $\left(s_{t+1}^{\text{ia}}\right)$
7: $\quad \Delta Q \leftarrow r_{t+1}^{\text{ia}} + \gamma \max_{a'} Q\left(s_{t+1}^{\text{ia}}, a'\right) - Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right)$
8: $\quad Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right) \leftarrow Q\left(s_t^{\text{ia}}, a_t^{\text{ia}}\right) + \alpha \Delta Q$
9: $\quad t \leftarrow t + t^g$

term "ϵ greedy" in line 4 means the action selected on the basis of the strategy that a random action is selected with probability ϵ; otherwise, an action a_t^{ia} that maximizes Q is selected (i.e., $\arg\max_{a'} Q\left(s_t^{\text{ia}}, a'\right)$) with probability $1 - \epsilon$. It indicates the epsilon-greedy algorithm and is to avoid convergence to a local optimum solution. The term "action" in line 5 shows the action of the instruction agent a_t^{ia}, which means the switch from the old control agent $\tilde{g}$ to the new control agent g, that is, $s_t^{\text{ia}} = \tilde{g}$ and $s_{t+1}^{\text{ia}} = g$. The term "agent learning (s_{t+1}^{ia})" in line 6 means control-agent learning (Algorithm 2). The control agent returns the maximum CEV during exploration and the number of exploration steps. Lines 7–8 show instruction-agent learning, which means that $Q\left(s^{\text{ia}}, a^{\text{ia}}\right)$ is updated from the relationship of state s^{ia}, action a^{ia}, and reward r^{ia}. ΔQ is called temporal difference error in RL, which indicates the difference between the current reward and the expected reward. At line 9, t^g means the number of iterations in Algorithm 2.

Control-agent algorithm

The control agent learns how to efficiently change the solution of the control engine. A state is defined as the solution of the control engine, action as the changing of the control solution of each control engine, and reward as the CEV (examples are given in Section 3.3.3).

The control-agent algorithm is shown in Algorithm 2. The control agent g is selected by the instruction agent, i.e., g corresponds to the current state

of instruction agent s^{ia}. Lines 1–2 show the initialization of each variable, and $\boldsymbol{A}^e$ means the initial solution corresponding to the control engine $\boldsymbol{e}$. The term "action" in line 5 means the changing part of the solution of the selected control engine $\boldsymbol{e}$. Then it outputs the result to the selected control engine $\boldsymbol{e}$. In line 6, the result is shared among other control engines through I/O-conversion unit. Lines 7–8 show the calculation of evaluation values of all control engines on the basis of the changed solution. The term "CEV calculation" in line 9 means the calculation the CEV as the reward of control agent r^g_{t+1}. The CEV is basically defined as follows:

$$\mathrm{CEV} = \sum_{e \in \boldsymbol{E}} \theta^e v^e_t, \tag{3.1}$$

where θ^e and v^e_t are the weighting parameter and evaluation value for the control engine $\boldsymbol{e}$, respectively. The term "end state" in line 12 means the termination condition of control-agent learning, i.e., the state that does not satisfy one or more constraints. That is, after reaching the solution that does not satisfy at least one constraint, the control agent stops the exploration. In lines 13 and 15, the control agent returns the maximum CEV during exploration as a reward for the instruction agent.

3.3 Use case of proposed method

This work considers the use cases where the extendable resource-integrated control method is required, i.e., where the control metrics and network control conditions are changed and added frequently. First, this work classified the general use case of NFV control using a combination of three elements: (1) control metric, (2) control objective, and (3) network model. This work also prepared four options as representatives of each element. One option is selected from (1) control metric, one option is selected from (2) control objective, and two options are selected from (3) network model. Finally, this work considers 12 use cases excluding four invalid combinations from the 16 ($= 2^4$) combinations.

In Sections 3.3.1 and 3.3.2, this work first describes the taxonomy of general use cases and the modeling of four options. Then this work describes the

Algorithm 2 Control-agent Learning.

initialize: $Q\left(s^g, a^g\right) \leftarrow 0$, for all s^g and a^g
initialize: $s_0^g \leftarrow \boldsymbol{A}^e$
for $t = 0$ to $T^g - 1$ **do**
    $a_t^g \leftarrow \epsilon$ greedy $\left(s_t^g\right)$
    $s_{t+1}^g \leftarrow$ action $\left(a_t^g\right)$
    $\boldsymbol{D}_{t+1} \leftarrow$ I/O conversion $\left(s_{t+1}^g\right)$
    for each $e \in \boldsymbol{E}$ **do**
        $v_{t+1}^e \leftarrow$ evaluation by each control engine $\left(\boldsymbol{D}_{t+1}\right)$
    $r_{t+1}^g \leftarrow$ CEV calculation $\left(\boldsymbol{V_{t+1}}, \boldsymbol{\theta}\right)$
    $\Delta Q \leftarrow r_{t+1}^g + \gamma \max_{a'} Q\left(s_{t+1}^g, a'\right) - Q\left(s_t^g, a_t^g\right)$
    $Q\left(s_t^g, a_t^g\right) \leftarrow Q\left(s_t^g, a_t^g\right) + \alpha \Delta Q$
    if s_{t+1}^g is *end state* **then**
        return $\max_{\tau \in \{0,1,\cdots,t\}} \left\{r_\tau^g\right\}, t+1$
    $t \leftarrow t+1$
return $\max_{\tau \in \{0,1,\cdots,T^g-1\}} \left\{r_\tau^g\right\}, T^g$

modeling and formulation of the proposed method and its extendable implementation in Sections 3.3.3 and 3.3.4, respectively.

Table 3.2: Summary of 12 types of use cases combining 4 options.

Options	1	2	3	4	5	6	7	8	9	10	11	12
(1) with IDS	✓	✓	✓	✓	✓	✓	✓	✓				
(2) with Reliability	✓	✓	✓	✓					✓	✓		
(3A) with Fixed node	✓	✓			✓	✓			✓		✓	
(3B) IDS isolation or sharing (✓: isolation)	✓		✓		✓		✓		–	–	–	–

3.3.1 Taxonomy of general use case

Several studies [7, 59–61] classify VNF/cloud resource control methods and their use cases. Various use cases are composed of a combination of 3 elements.

(1) Control metric: The control metric is defined by parameters to characterize the state of a controlled network, e.g., VNF types, VNF model (e.g., CPU, memory, and storage), VNF placements, the combination of the VNFs, the order to go through the VNFs and routes between the VNFs, etc. Specifically, each control metric determines the constraints, e.g., link bandwidth, latency, server capacity, the maximum number of chaining VNFs, etc.

(2) Control objective: Control objectives can be categorized as follows: improvement of resources utilization efficiency (e.g., link and server), network performance (e.g., traffic throughput and latency), quality of service/quality of experience (QoS/QoE), an acceptance rate of service demands, energy efficiency, security and reliability, etc. The number of control objectives also depends on the use case. The use cases in NFV often introduce multiple control objectives. It has been reported that 34% of the previous studies on VM placement used the multi-objective approach [59].

(3) Network model: The network model is a specific representation of a controlled network and user demands depending on each use case. The example of network model element is as follows: network topology, traffic transport rule (e.g., route splittable or not), node placement rule (e.g., fixed node placement or not), and resource isolation rule (e.g., with or without network slicing), etc.

This work describes two cases with different network models as an example. One example case is when assuming the service function chaining (SFC) in a TSP network. In this case, this work generally assumes the communication between the client as an origin node and the server as middle nodes or a destination node. Then, this work models that the client node is fixed because its location such as a company building using an SFC service is predetermined by the client's location, and the server node can migrate because that function is virtualized as a VM or VNF. In this model, the server resources to be allocated to individual VNFs are separated among users for reasons such as the VNF license fee and security. The other example case is when assuming the data transportation in an inter-DC network. In this case, it is assumed that the

functions in origin, middle, and destination nodes can migrate because these functions are virtualized as VMs or VNFs. When a user requests multiple VN demands, or when the DC operator or TSP manages all VN demands, the server resources to be allocated to individual VNFs can be shared among VN demands. The maximum numbers of VNFs and concurrent sessions are practically limited due to the constraints of license and cost. Therefore, it can be modeled that one VNF allocated to near the origin node is selected until the maximum number of concurrent sessions is reached.

3.3.2 Modeling of use cases and options

This work selects 4 options from the above elements to evaluate the proposed method's extendability and coverage for various use cases. Each option is **(1) with IDS** (i.e., a representative example of adding a control metric), **(2) with Reliability** (i.e., a representative example of adding a control objective), **(3A) with Fixed node** and **(3B) IDS isolation or sharing** (i.e., representative examples of changing network models). Table 3.2 shows 12 use cases combining 4 options. There are only 12 use cases because the (3B) IDS isolation or sharing option is effective only under (1) with the IDS option.

This work first describes the condition of the simplest case (Case #12). In this case, this work assumes the use case of computing resource optimization in a single DC as an example. The origin and destination nodes are VMs, i.e., both nodes can migrate. The control metrics are routes and VM placements. Link capacity and server capacity are imposed as constraints. Maximum link utilization efficiency and maximum server utilization efficiency are introduced as control objectives. All routes between the origin and destination are splittable. That is, the traffic between an origin–destination (OD) node pair can be split into multiple routes.

Next, this work describes the condition of each option. The **(1) with IDS** option adds IDS placements as control metrics. It is an option passing through an IDS between the origin and destination for all user demands. The same as for VM placements, server capacity is imposed as a constraint and maximizing server utilization efficiency is introduced as a control objective for IDS placements. The **(2) with Reliability** option adds maximizing total

reliability as a control objective. In this study, reliability is defined by the probability that a packet can go between two points. In other words, it is defined by the one minus failure probability. When the route of each OD is splitting, the reliability of each OD is calculated by multiplying the split ratio and reliability of each route. The formulation of total reliability is described in Section 3.3.3. The **(3A) with Fixed node** option decides whether the origin and/or destination node is a fixed node or can migrate. An example of a fixed node is a client node. The **(3B) IDS isolation or sharing** option decides whether IDS resources are shared among VNs or not. When isolating IDSs among VNs, the number of IDSs (N_{ids}) that need to be allocated is the same as the number of VNs (N_{VN}), that is, $N_{\mathrm{ids}} = N_{\mathrm{VN}}$. When sharing IDSs among VNs, the N_{ids} is less than the N_{VN}, that is, $N_{\mathrm{ids}} = M < N_{\mathrm{VN}}$.

Some previous studies can be classified into 12 use cases. The policy and VM consolidation method in cloud DC [6] are similar to Cases #5–#8. The disaster avoidance control method in a TSP network [62,63] is similar to Cases #9–#10. The joint VM placement and routing control method in DC [10] is similar to Cases #11–#12. However, to the best of our knowledge, no method has been developed that corresponds to Cases #1–#4. In addition, **no method has been developed that can handle all cases with one extendable algorithm.**

3.3.3 Modeling of proposed method

This work describes the modeling and formulation of the proposed method on the basis of Case #1 since it is redundant to explain the modeling of 12 use cases one by one. This work considers the use case in which this work provides a secure and reliable cloud-computing service consisting of routes between VMs via an IDS. This work describes each formulation of the algorithm and modeling of the proposed method. In this case, the control metrics are routes, VM placements, IDS placements, and reliability. Each control algorithm is pre-formulated. The symbols used in the formulation are defined in Table 3.3.

Table 3.3: Symbol descriptions for control engines.

Symbols	**Definitions**
N_{server}	Number of servers
$\boldsymbol{N}, \boldsymbol{S}, \boldsymbol{L}$	Node set, server set, link set
$P(\boldsymbol{N}, \boldsymbol{L}) = P(\boldsymbol{S}, \boldsymbol{L})$	Physical Network graph
link $(i, j) \in \boldsymbol{L}$	Link from node i to node j
c_{ij}^{link}	Link capacity of link (i, j)
c_i^{server}	i^{th} server capacity
N_{VN}	Number of VNs
$N_{\text{cli}}, N_{\text{vm}}, N_{\text{ids}}$	Number of clients, VMs, and IDSs
$\boldsymbol{C}, \boldsymbol{V}, \boldsymbol{I}$	Client set, VM set, IDS set
c_i^{ids}	i^{th} IDS capacity
w_i^{vm}, w_j^{ids}	i^{th} VM size, j^{th} IDS size
t_i^{VN}	OD Traffic demands for i^{th} VN
$\boldsymbol{\Xi}^{\text{cli}} := \left\{\xi_{ij}^{\text{cli}}\right\}$	Client node placement (i^{th} client, j^{th} node)
$\boldsymbol{T}^{\text{node}} := \{t_{pq}\}$	Traffic from node p to node q
$\boldsymbol{T}^{\text{vm}} := \left\{t_{ij}^{\text{vm}}\right\}$	Traffic from VM i to VM j
x_{ij}^{pq}	Proportion of passed t_{pq} on link (i, j)
$U_{\text{max}}^{\text{link}}$	Maximum link utilization
$\boldsymbol{\Xi}^{\text{vm}} := \left\{\xi_{ij}^{\text{vm}}\right\}$	VM allocation (i^{th} VM, j^{th} server)
$\boldsymbol{\Xi}^{\text{ids}} := \left\{\xi_{ij}^{\text{ids}}\right\}$	IDS allocation (i^{th} IDS, j^{th} server)
$U_{\text{max}}^{\text{server}}$	Maximum server utilization
r_{ij}^{link}	Link reliability of link (i, j)
r_i^{node}	i^{th} node reliability
R^{total}	Total reliability

Network

This work assumes that each physical server is connected to each node, that is $P(\boldsymbol{N}, \boldsymbol{L}) = P(\boldsymbol{S}, \boldsymbol{L})$. When each VN request is accepted, the amounts of server and link resources consumed depend on the request size.

This work assumes that there is a certain number of VN requests N_{vn}. A VN request consists of one origin (i.e., client) and one destination (i.e., VM),

OD traffic demands, and VM size. Each VM is allocated to a physical server. The VM size indicates the processing capacity of the VM request, such as the requested number of CPU cores. This work also assumes that each OD traffic demand is routed through an IDS, which is also allocated to a physical server. The IDS size also indicates the processing capacity. Note that if the client and IDS for an OD pair are allocated in the same node, the OD traffic demand between the client and IDS on the network is regarded as 0. Similarly, if VM and IDS for an OD pair are allocated in the same server, the OD traffic demand between them is regarded as 0.

Control algorithms

This work introduces four control engines: route, VM, IDS, and reliability ($\boldsymbol{E} = \{\text{Route}, \text{VM}, \text{IDS}, \text{Reliability}\}$). All control engines have pre-specified control algorithms. The calculation procedure of the initial solution is as follows. After the VM and IDS control algorithms calculate the optimal allocations without taking into account the constraints of other control algorithms, the route control algorithm calculates the end-to-end route between VMs via an IDS. Finally, the reliability control algorithm calculates the reliability on the basis of all end-to-end routes.

This work introduces three objective functions: minimization of maximum link utilization for route control, minimization of maximum server utilization for VM and IDS controls, and maximization of total reliability for reliability control. This work imposes three constraints: link capacity for route control, server capacity for VM control, and server capacity for IDS control.

The route control algorithm is formulated as follows:

$$\min \quad : \quad U_{\max}^{\text{link}} \tag{3.2}$$

$$\text{s.t.} \quad : \quad \sum_{j:(i,j)\in L} x_{ij}^{pq} - \sum_{j:(j,i)\in L} x_{ji}^{pq} = 0 \tag{3.3}$$

$$(\forall p, q \in N, i \neq p, i \neq q)$$

$$\sum_{j:(i,j)\in L} x_{ij}^{pq} - \sum_{j:(i,j)\in L} x_{ji}^{pq} = 1 \tag{3.4}$$

$$(\forall p, q \in N, i = p)$$

$$\sum_{p,q\in N} t_{pq} x_{ij}^{pq} \leq c_{ij}^{\text{link}} U_{\max}^{\text{link}}$$

$$(\forall (i,j) \in L, \forall p,q \in N) \tag{3.5}$$

$$0 \leq x_{ij}^{pq} \leq 1 \qquad (\forall (i,j) \in L, \forall p,q \in N) \tag{3.6}$$

$$0 \leq U_{\max}^{\text{link}} \leq 1. \tag{3.7}$$

This algorithm calculates a routing variable x_{ij}^{pq} to minimize the link utilization $U_{\max}^{\text{link}}$ while satisfying the constraints in (3)–(7), where x_{ij}^{pq} shows the proportion of passing OD traffic demands t_{pq} on the link (i,j). Equations (3)–(4) show the traffic flow conservation law. Equation (5) shows the constraint of link capacity. Equations (6)–(7) show the range of variables.

The VM control algorithm is formulated as follows:

$$\text{min} \quad : \quad U_{\max}^{\text{server}} \tag{3.8}$$

$$\text{s.t.} \quad : \quad \sum_{s_k \in S} \xi_{ik}^{\text{vm}} = 1 \qquad (\forall v_i \in V) \tag{3.9}$$

$$\sum_{v_i \in V} w_i^{\text{vm}} \xi_{ik}^{\text{vm}} \leq c_k^{\text{server}} U_{\max}^{\text{server}} \qquad (\forall s_k \in S) \tag{3.10}$$

$$\xi_{ik}^{\text{vm}} \in \{0,1\} \tag{3.11}$$

$$0 \leq U_{\max}^{\text{server}} \leq 1. \tag{3.12}$$

This algorithm calculates an VM allocation variable ξ_{ik}^{vm} to minimize the server utilization $U_{\max}^{\text{server}}$ while satisfying the constraints in (8)–(12), where ξ_{ik}^{vm} shows the VM solution in which ξ_{ik}^{vm} is 1 if i^{th} VM is assigned to the k^{th} server; otherwise, 0. Equation (9) shows the VM conservation law. In other words, it shows that each VM must be allocated to any server. Equation (10) shows the constraint of server capacity. Equations (11)–(12) show the range of variables.

The formulation of the IDS control algorithm replaces w_i^{vm} and ξ_{ik}^{vm} with w_j^{ids} and ξ_{jk}^{ids} for that of the VM control algorithm. Similarly, IDS allocation ξ_{jk}^{ids} indicates the IDS solution in which ξ_{jk}^{ids} is 1 if j^{th} IDS is assigned to the k^{th} server; otherwise, 0.

In this study, reliability is defined by the probability that a packet can go between two points. Especially, node reliability is defined by the packet reachable probability from node ingress to node egress. In other words, it is defined by the one minus node failure probability. The link reliability is also similar. The reliability between ODs is defined as the product of each reliability going through each node and each link between ODs. The reliability control

algorithm is formulated as follows:

$$\min \quad : \quad R^{\text{total}} \tag{3.13}$$

$$R^{\text{total}} = \frac{1}{\sum_{k=1}^{N_{\text{VN}}} t_k^{\text{VN}}} \sum_{k=1}^{N_{\text{VN}}} t_k^{\text{VN}} R_k^{\text{VN}} \tag{3.14}$$

$$R_k^{\text{VN}} = \sum_{p \in \text{path}(k)} r_p \left(\prod_{i \in N_p} r_i^{\text{node}} \prod_{(i,j) \in L_p} r_{ij}^{\text{link}} \right). \tag{3.15}$$

As shown in (3.14), total reliability R^{total} is defined as the weighted average of each VN reliability R_k^{VN}, where the weight is determined by the traffic demand to each VN, i.e., $\sum_{k=1}^{N_{\text{VN}}} t_k^{\text{VN}}$. Here, R_k^{VN} is calculated as shown in (3.15). This work explains this formula. The terms r_i^{node} and r_{ij}^{link} indicate the reliability of node i and the link between nodes i and j. If each VN has multiple paths, each VN's reliability is calculated using the traffic splitting ratio. For each path $p \in \text{path}(k)$ of the k^{th} VN, the traffic splitting ratio r_p, the set of nodes that the path p passes through N_p, and the set of links that the path p passes through L_p are defined. The above three parameters are calculated by each VN allocation result, each traffic demand $\boldsymbol{T}^{\text{node}}$, and the route control engine's solution x_{ij}^{pq}.

Note that this work selected these control algorithms as examples, which are commonly used in previous studies. In the proposed method, each control algorithm is saved as a model file and can be changed by only changing the model file.

Coordination algorithm

This work introduces VM and IDS control agents as the control agents ($\boldsymbol{G} = \{\text{VM}, \text{IDS}\}$). The route control agent is not introduced here because any routes between the VNFs are not changed unless the VNF placements change. The route control engine is used only to calculate the part of the CEV by solving the route control algorithm in each step. Similarly, the reliability control agent is not introduced.

The state of a control agent defines the VM or IDS allocation, that is, $s^{\text{VM}} = \boldsymbol{\Xi}^{\text{vm}}$ or $s^{\text{IDS}} = \boldsymbol{\Xi}^{\text{ids}}$. The action of the control agent defines one VM or IDS migration. The VM or IDS to migrate is selected from the most used

server, and the destination server is selected on the basis of its agent strategy. Note that, in this action, only one migration is executed, and the VNF control algorithm described using (3.8)–(3.12) is not solved. The reward of the control agent defines the CEV if all constraints are satisfied; otherwise, the penalty is −100. The CEV is defined as follows:

$$r_t^g = \theta^{\text{link}} \left(1 - U_{\max}^{\text{link}}\right) + \theta^{\text{server}} \left(1 - \tilde{U}_{\max}^{\text{server}}\right) + \theta^{\text{r}} R_t^{\text{total}}, \tag{3.16}$$

where θ^{link} and θ^{server}, and θ^{r} are weighting parameters indicating the importance of each control-objective function. The term $\tilde{U}_{\max}^{\text{server}}$ is the maximum server utilization after aggregating VM and IDS allocations. The I/O-conversion unit calculates VN allocation results, which are the set of the origin node (or clients placement), middle node (allocated IDS placement), destination server (allocated VM placement), and $\boldsymbol{T}^{\text{node}}$, on the basis of $\boldsymbol{\Xi}^{\text{vm}}$, $\boldsymbol{\Xi}^{\text{ids}}$, $\boldsymbol{\Xi}^{\text{cli}}$, and t_i^{VN}.

3.3.4 Implementation difference between options

This work describes the implementation differences with and without each option shown in Table 3.2. This work describes the required additional implementation in comparison with the situation where the implementation of proposed methods for Case #12 is completed. This work also indicates that the above implementation can easily be completed.

(1) with IDS: When the option is added, this work needs to introduce an IDS agent and IDS control engine. The RL algorithm and its modeling of state, action, and reward are the same for the VM agent and IDS agent, so no additional implementation of the Python code is required for adding the IDS agent. Similarly, no additional implementation of the code is required for adding IDS control engine because the objective function and constraints of the IDS engine are the same as those for the VM control engine in this use case. Even if the formulation of IDS control engine is changed, the formulation of that engine is modularized as a file that describes optimization problem formulations, so the engine can be reformulated by changing a few lines of that file.

In I/O-conversion unit calculation, the format of VN allocation results is changed to that of adding the middle server node information. In the CEV

calculation, the term $U_{\max}^{\text{server}}$ of the maximum server utilization is changed to $\tilde{U}_{\max}^{\text{server}}$, which is that of the value after aggregating VM and IDS allocations. In this implementation, the above change can be developed with a change of about 10 lines of Python code.

(2) with Reliability: When the option adds, this work needs to introduce Reliability control algorithms. Though the implementation of the engine is newly required, it is extendable because the existing code does not change. In addition, the formula to calculate CEV slightly needs to be changed from (3.17) to (3.16).

$$r_t^g = \theta^{\text{link}} \left(1 - U_{\max}^{\text{link}}\right) + \theta^{\text{server}} \left(1 - U_{\max}^{\text{server}}\right) \tag{3.17}$$

(3A) with Fixed node: When the option changes, this work needs to slightly modify the I/O-conversion unit implementation. In the I/O-conversion unit calculation, origin node placement returns from $\boldsymbol{\Xi}^{\text{cli}}$ if with the Fixed node option, otherwise, it returns $\boldsymbol{\Xi}^{\text{vm}}$.

(3B) IDS isolation or sharing: When the option changes, this work needs to modify the I/O-conversion unit implementation. When the IDS isolation condition is used, the I/O-conversion unit returns the specific IDS for each VN demand, that is, the i^{th} IDS is exclusively allocated to the i^{th} VN. If IDS sharing condition, the I/O-conversion unit returns the best IDS selected from IDS set $\boldsymbol{I}$. The best IDS is defined as the IDS that satisfies two conditions: the length of total OD route via IDS is the shortest, and concurrent sessions are less than the IDS capacity c_i^{ids}. In this implementation, the above change can be developed with a change of about 10 lines of Python code.

3.4 Evaluation

This work evaluated the effectiveness of the proposed algorithm through simulations in terms of solution-exploration speed, difference from the optimal solution, scalability, and extendability. This work uses the use cases in Section 3.3 to evaluate the extendable resource-integrated control method.

This work first evaluates the solution-exploration speed to assess whether the proposed method can find the solution with improved CEV within the practical iterations because the proposed method generally seems to need more

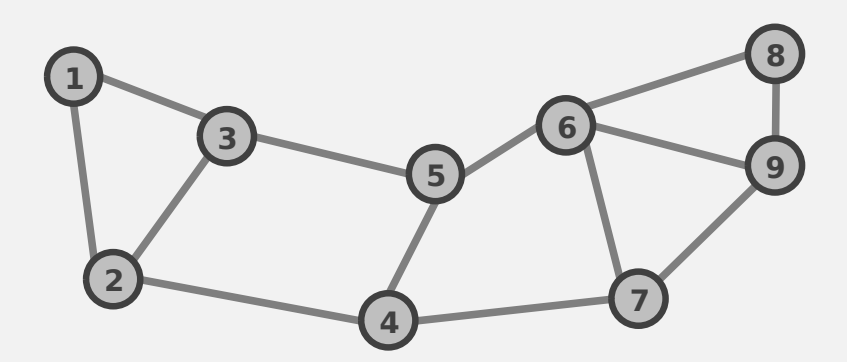


Figure 3.3: Internet2 topology. (©2020 IEICE.)

iterations than the combined approach developed for a specific problem. In addition, this work assesses whether RL can find better solution efficiently. Then, this work also investigates the difference from the optimal solution. After that, this work evaluates the scalability of the proposed method to estimate the practical range of N_{VN} where the CEV can be improved within the practical computational time. In addition, this work also discusses the extendability of the proposed method. Since the extendability of the proposed method is difficult to evaluate quantitatively, this work shows that the proposed method makes it possible to solve all use cases. Since it is redundant to discuss all 12 results, this work focused on the 6 use cases for which the effects of changing each option need to be discussed (Cases #1, #3, #4, #5, #8, #12). Then, this work investigates the differences between the proposed coordinated method and the previous combined method and also discusses how easy/difficult to build and solve the problem. Finally, this work discusses the applicability of the proposed method.

3.4.1 Evaluation conditions

For the physical network conditions, this work used the topology of Internet2 [64], which consists of 9 nodes (Fig. 3.3). In particular, this work assumes a local disaster near node 6 and sets $r_6^{\mathrm{node}} = 0.9$ and $r_{67}^{\mathrm{link}} = r_{69}^{\mathrm{link}} = 0.5$. Other r_{ij}^{link} and r_i^{node} are set to 1.0. For the VN demand conditions, the location of each client node is randomly generated. Moreover, the OD traffic demand t_i^{VN} is ran-

domly generated within the range of 0–1.0 Gbps so as to arrange the average as 0.5 Gbps. Each VM size is randomly given an integer value, and each IDS size is fixed to an integer value. The average server utilization is set to 80% by changing each server capacity proportionally to the N^{VN} and slightly adjusting the VM size. For the agent conditions, this work sets the total exploration steps of instruction and control agents to $T = 5000$, and $T^g = 20$.

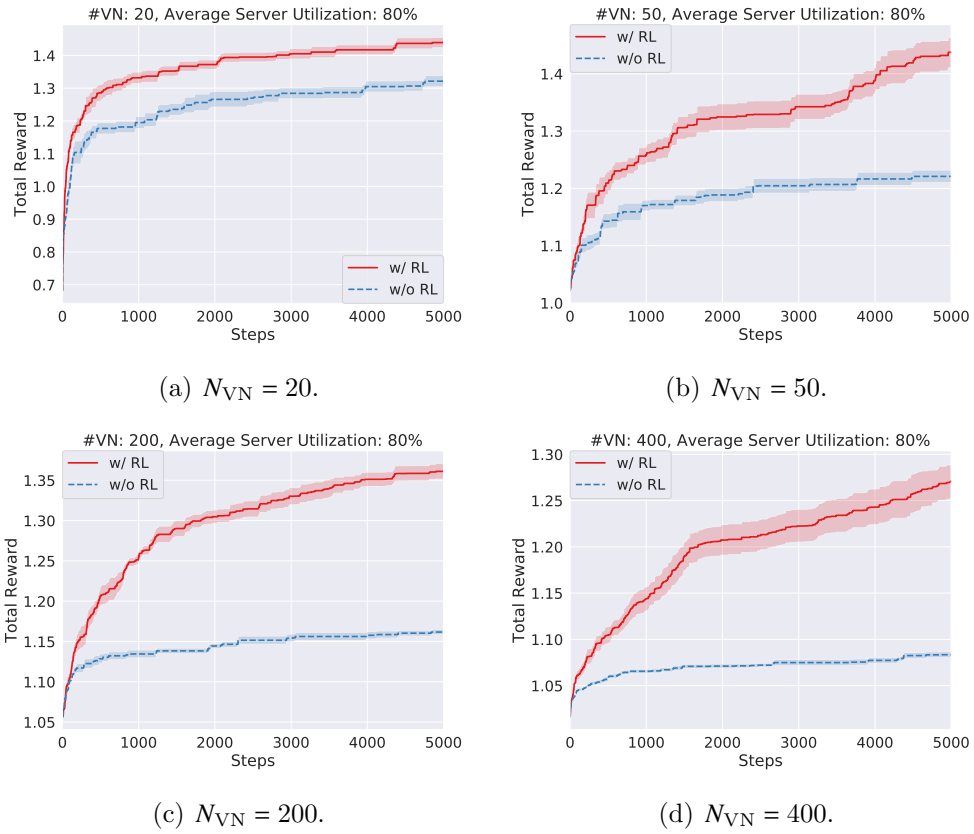
In above conditions, this work varied N_{VN} from 20 to 2000 and varied use-cases from 1 to 12. Some parameters increase proportionally as shown in Table 3.4 as the number of VNs are increased from 20 to 2000. Some parameters also set depending on the selected use-cases as shown in Table 3.6. In addition, this work excluded the cases in each of which an initial solution does not satisfy all constraints in all evaluations. This is because starting from an unsatisfied initial solution would drastically decrease the performance of the solution. The way to find a feasible initial solution is discussed in Section 3.4.2, which is for future study.

3.4.2 Evaluation results

This work implemented the proposed coordination algorithm and physical network simulator using Python from scratch and each pre-specified control algorithm using the GNU Linear Programming Kit (GLPK) [65] to calculate initial solutions.

Table 3.4: Scale parameters for Case #1.

Definitions		20	50	200	400	800	1000	1500	2000
Number of VNs	N_{VN}	20	50	200	400	800	1000	1500	2000
Number of VMs	N_{vm}	20	50	200	400	800	1000	1500	2000
Number of IDSs	N_{ids}	20	50	200	400	800	1000	1500	2000
Number of clients	N_{cli}	20	50	200	400	800	1000	1500	2000
i^{th} server capacity	c_i^{server}	12–14	30–36	120–144	240–288	480–576	600–720	900–1080	1200–1400
Link capacity of link (i, j)	c_{ij}^{link}	3	7.5	30	60	120	150	225	300



(a) $N_{\mathrm{VN}} = 20$.

(b) $N_{\mathrm{VN}} = 50$.

(c) $N_{\mathrm{VN}} = 200$.

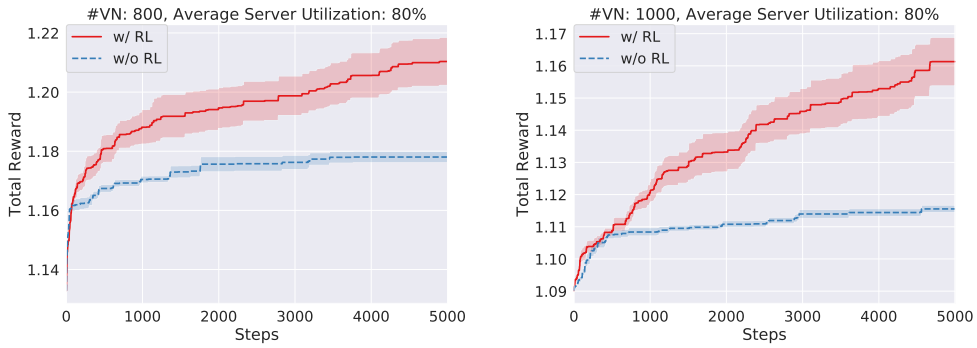
(d) $N_{\mathrm{VN}} = 400$.

Figure 3.4: Solution-exploration speed for Case #1 and its N_{VN} dependency (1). (©2020 IEICE.)

Discussion on solution-exploration speed

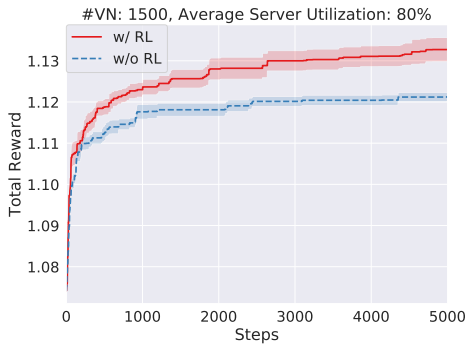
This work compared the solution-exploration speeds of the proposed algorithm based on RL (w/ RL) and an algorithm based on changing solutions randomly (w/o RL). Note that, in the case of w/o RL, this work sets $\epsilon = 1$ and also skipped both agent learning steps, i.e., lines 7–8 in Algorithm 1 and lines 10–11 in Algorithm 2. In this evaluation, this work uses Case #1 and sets to each weighting parameter $\theta^{\mathrm{link}} = \theta^{\mathrm{server}} = \theta^{\mathrm{r}} = 1$.

This work first discusses the case when $N^{\mathrm{VN}} = 200$ as a baseline. This work will discuss other Figs. 3.4(a)–3.5(d) in Sections 3.4.2 and 3.4.2. Figure 3.4(c) shows the solution-exploration speeds for Case #1, which is the average transition of the best CEV, which is defined by the highest CEV found until the current exploring step. Note that each time CEV is defined by the total reward

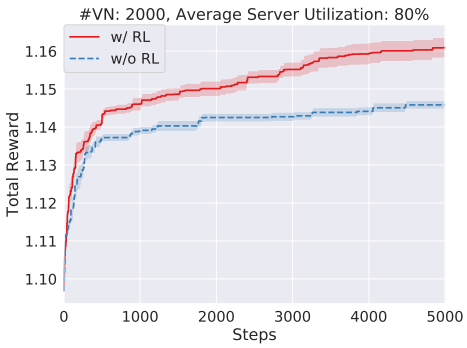


(a) $N_{\mathrm{VN}} = 800$.

(b) $N_{\mathrm{VN}} = 1000$.



(c) $N_{\mathrm{VN}} = 1500$.

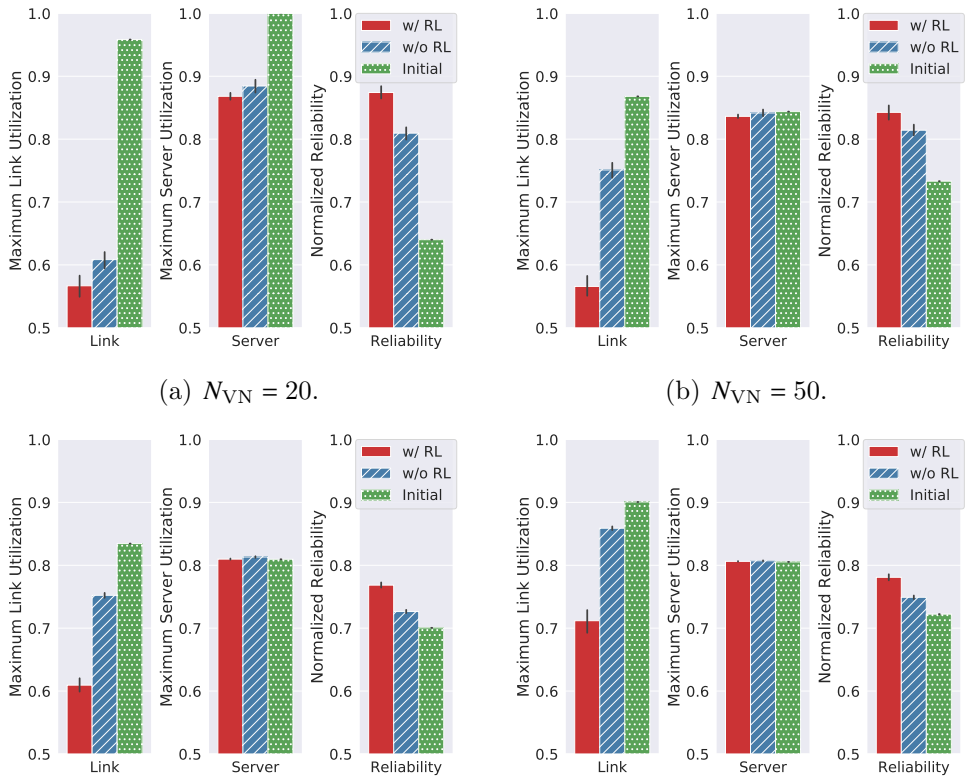


(d) $N_{\mathrm{VN}} = 2000$.

Figure 3.5: Solution-exploration speed for Case #1 and its N_{VN} dependency (2). (©2020 IEICE.)

r_t^g shown in (3.16). This work carried out 10 calculations with a fixed initial solution. The width of each line indicates the standard deviation ($\pm\sigma$). Though the initial CEV was low due to the control conflict mentioned in Section 3.1, the best CEV was improved by repeating the exploration in both cases w/ RL and w/o RL. Results of the comparison between w/ and w/o RL in Fig. 3.4(c) indicated that RL could find a better CEV solution within 5000 exploration steps. The reason is that the agent of RL learns the strategy for how to find better allocations efficiently from past exploration steps.

Figure 3.6(c) shows the components of each objective function value in the best solution for Case #1. Initial in Fig. 3.6(c) means above values for the initial solution. w/ RL improves the total reward about 0.3 in Fig. 3.4(c), which is equivalent to improving the sum of link utilization, server utilization,

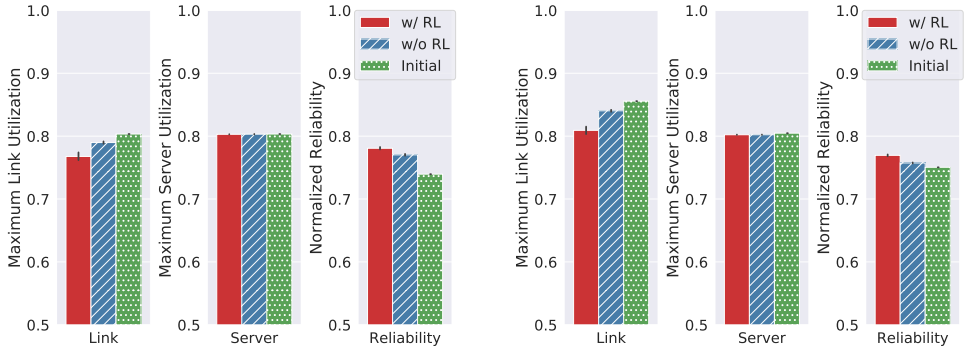


(a) $N_{\mathrm{VN}} = 20$. (b) $N_{\mathrm{VN}} = 50$.

(c) $N_{\mathrm{VN}} = 200$. (d) $N_{\mathrm{VN}} = 400$.

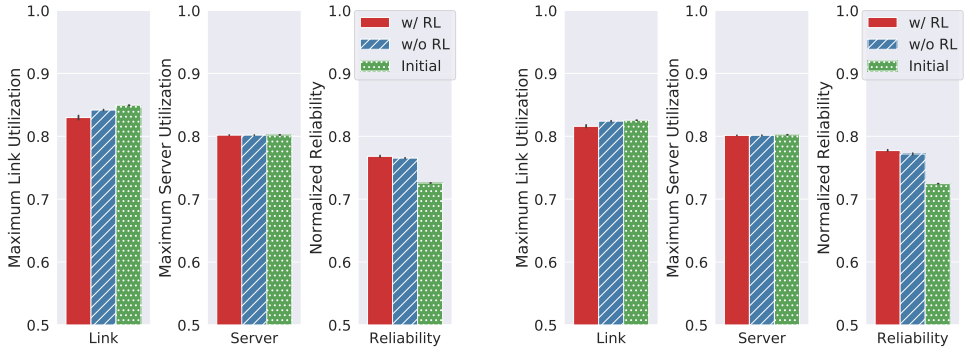
Figure 3.6: Components of each objective function value in the best solution for Case #1 and its N_{VN} dependency (1). (©2020 IEICE.)

and reliability by 30%. Since this work assumed $\theta^{\mathrm{link}} = \theta^{\mathrm{server}} = \theta^{\mathrm{r}} = 1$ in this evaluation, an increase of 0.01 for the CEV is equivalent to a 1% improvement in the sum of link utilization, server utilization, and reliability. Of the 30% total improvement, the maximum link utilization improvement is about 22% and the total reliability improvement is about 8%. Note that, since the average server utilization is set to 80% in all evaluations, the optimal value of the maximum server utilization is 80%. In addition, since an initial solution of VM and IDS placements is calculated by the VM and IDS control engines for minimizing maximum server utilization, this value of the initial solution is near to 80%.



(a) $N_{\mathrm{VN}} = 800$.

(b) $N_{\mathrm{VN}} = 1000$.



(c) $N_{\mathrm{VN}} = 1500$.

(d) $N_{\mathrm{VN}} = 2000$.

Figure 3.7: Components of each objective function value in the best solution for Case #1 and its N_{VN} dependency (2). (©2020 IEICE.)

Discussion on difference to optimal solution

This work discusses the difference to the optimal solution in the case of w/ RL and Case #1. General VN allocation problems are known to be NP-hard [66]. In addition, there are no previous studies for calculating an optimal solution of Case #1 without approximation. On the other hand, the policy value $Q(s, a)$ of RL has been analytically proved to converge to the optimal policy $Q^*(s, a)$ in an infinite number of exploration steps by a policy improvement theorem [67]. Therefore, when increasing the total exploration steps T to infinity, the solution and its CEV absolutely converge to the optimal solution and optimal value. Since infinite iterations are impossible, this work regarded a sub-optimal solution/CEV as the converged solution/CEV when the number of exploration steps sufficiently increased.

Table 3.5: CEV convergence ratio when w/ RL and $N_{\mathrm{VN}} = 200$.

Steps	Best CEV	Convergence ratio
0 (Initial)	1.06	0.00
5.0×10^3	1.35	0.56
1.0×10^4	1.38	0.62
5.0×10^4	1.48	0.82
1.0×10^5	1.52	0.89
5.0×10^5	1.57	0.99
1.0×10^6	1.58	1.00
1.5×10^6	1.58	1.00

Table 3.5 shows the convergence speed to the sub-optimal CEV when $N_{\mathrm{NV}} = 200$. This evaluation corresponds to the case where the number of exploration steps was increased for the evaluation in Fig. 3.4(c). Since the best CEV sufficiently converges when the total exploration steps T are increased to 1.5×10^6, CEV = 1.58 is regarded as the sub-optimal CEV and the solution at the time is regarded as the sub-optimal solution. The convergence ratio is defined as the best CEV minus initial CEV divided by sub-optimal CEV minus initial CEV. This work defined sufficient converge as the case when the error of the convergence ratio has converged to 1% or less. As shown in Table 3.5, the convergence ratio reaches 56% of the sub-optimal solution in 5000 steps and 82% of the sub-optimal solution in 50,000 steps. This convergence speed seems to suffice as a general NP-hard problem solution.

Discussion on scalability

Figures 3.4–3.7 show the solution-exploration speed and the components of each objective function value in the best solution when this work varied N_{VN} from 20 to 2000 for Case #1. It reveals that the proposed method can improve the solution by repeating the exploration in all cases of both algorithms (w/ RL and w/o RL). In addition, RL can more efficiently explore better solutions than w/o RL. Note that the performance of the initial solution depends on the randomness of the initial OD traffic and initial client node, so it cannot be compared uniformly in each case.

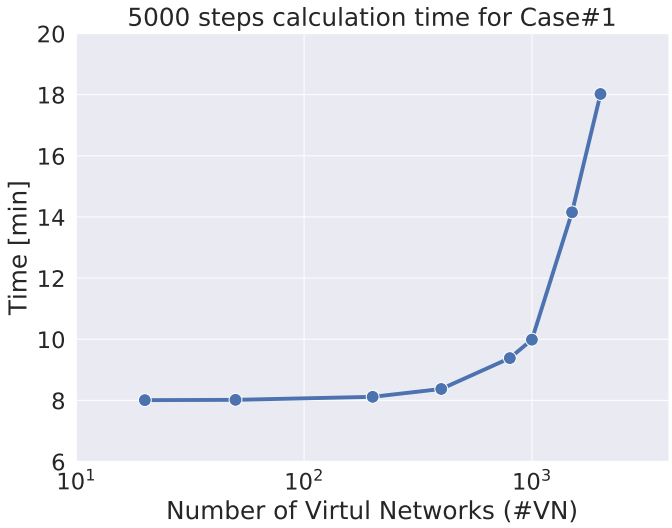


Figure 3.8: Computation time. (©2020 IEICE.)

In the case of N_{VN} is 200, the improvement of the solution is about 30% for w/ RL and about 10% for w/o RL. On the other hand, when the $N_{\mathrm{VN}} = 20$ and $N_{\mathrm{VN}} = 1500$ or more, the improvement of the solution is reduced to 10% or less. This shows that the improvement of the solution basically decreases as the N_{VN} increases except for the case of $N_{\mathrm{VN}} = 20$. The performance decreased in the case of $N_{\mathrm{VN}} = 20$ because a better solution was found easily even w/o RL since the solution exploration space is sufficiently small. The performance decreased in the case of $N_{\mathrm{VN}} = 1500$ or more because the learning of RL is not sufficient due to the number of total exploration steps close to the N_{VN}. From the above discussion, this work concludes that the proposed method is effective in the range of $N_{\mathrm{VN}} = 50$ to 1000.

Figure 3.8 shows the computation time of the proposed algorithm (w/ RL) for N_{VN} 20 and 2000. The calculations were performed on a Intel core i7 4790k CPU of a single core. The computation time increases depending on the number of steps proportionally. Although N_{VN} increased 100 times, the computation time increased only several times. This means that, from the viewpoint of computation time, the proposed method is scalable with respect to N_{VN}, with up to 1000 VNs. Note that w/o RL has almost the same computation time as w/ RL. The difference between w/ RL and w/o RL is the

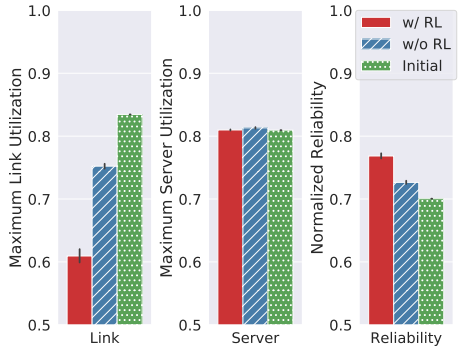
overhead time of RL, which is less than 1% of the total computation time.

Figure 3.8 also shows that the calculation time of the proposed method until 5000 steps is less than 10 minutes in the range up to 1000 VNs. This work thus considered that the calculation time allows enough practice. In NFV environments, each VN demand is statistically multiplexed by multiple users sharing the VN. For that reason, the proposed control system mainly targets static VN demand allocation, which considers VN demands to be fixed within a particular period (e.g., more than one hour). Moreover, the proposed method can adjust the calculation time to modify the number of total exploration steps in accordance with the required calculation time.

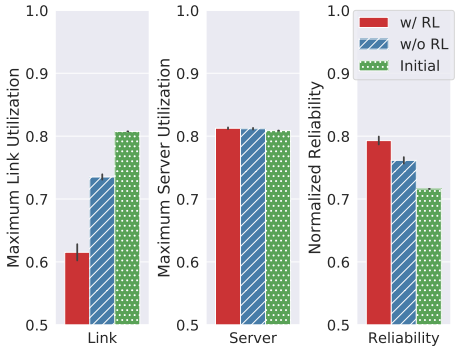
The computation time of the proposed method is determined by the calculation time of each control engine. In this evaluation, the route control engine is formulated as a linear programming (LP) problem, and its calculation time is less than one second. However, if the route control engine is formulated as an integer linear programming (ILP) problem, e.g., non-split route case and path-base route control case, the computation time of the proposed method will increase dramatically. When each exploration step contains ILP problems, the following solution seems to be effective: set the upper limit for calculation time of each step, approximate by limiting of route candidates, and use the heuristic method for route calculation.

Table 3.6: Parameters depend on each case when N_{VN} is 200.

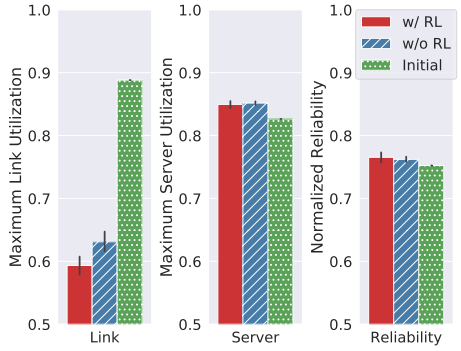
Definitions		1	2	3	4	5	6	7	8	9	10	11	12
Number of IDSs	N_{ids}	200	10	200	10	200	10	200	10	0	0	0	0
Number of clients	N_{cli}	200	200	0	0	200	200	0	0	200	0	200	0
i^{th} VM size	w_i^{vm}	1–3	1–3	1–3	1–3	1–3	1–3	1–3	1–3	3–8	3–8	3–8	3–8
i^{th} IDS size	w_i^{ids}	2	40	2	40	2	40	2	40	0	0	0	0
i^{th} IDS capacity	c_i^{ids}	1	20	1	20	1	20	1	20	0	0	0	0



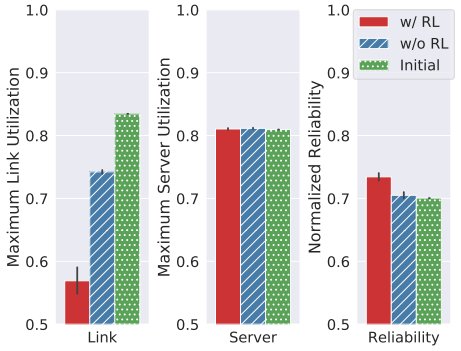
(a) Case #1. (w/ IDS, w/ Reliability, w/ Fixed node, IDS isolation.)



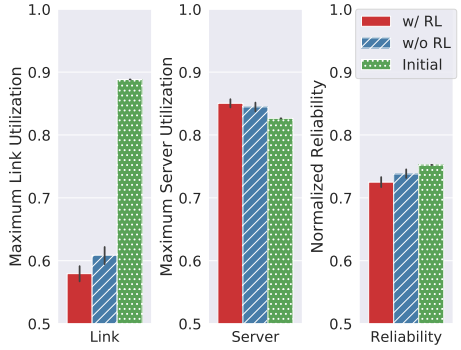
(b) Case #3. (w/ IDS, w/ Reliability, IDS isolation.)



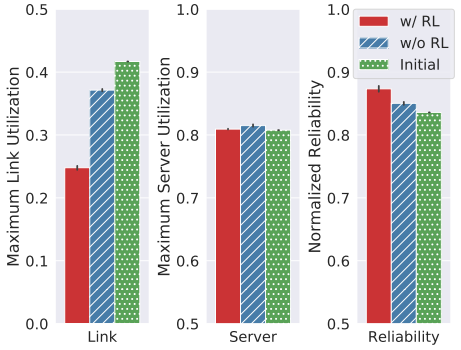
(c) Case #4. (w/ IDS, w/ Reliability, IDS sharing.)



(d) Case #5. (w/ IDS, w/ Fixed node, IDS isolation.)

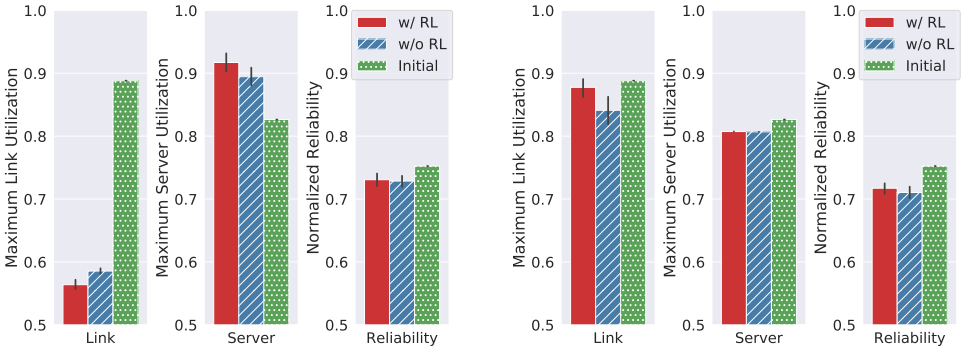


(e) Case #8. (w/ IDS, IDS sharing.)



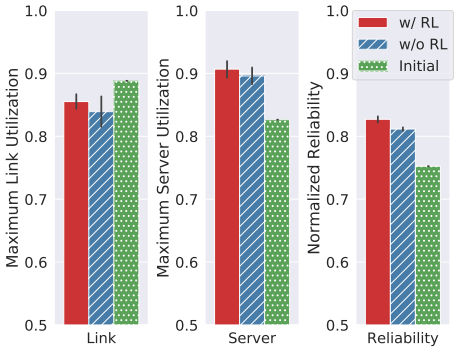
(f) Case #12. (default.)

Figure 3.9: Components of each objective function value in the best solution for each case. (©2020 IEICE.)



(a) $(\theta^{\text{link}}, \theta^{\text{server}}, \theta^{\text{r}}) = (10, 0, 0)$.

(b) $(\theta^{\text{link}}, \theta^{\text{server}}, \theta^{\text{r}}) = (0, 10, 0)$.



(c) $(\theta^{\text{link}}, \theta^{\text{server}}, \theta^{\text{r}}) = (0, 0, 10)$.

Figure 3.10: Weighting parameter dependency of components of each objective function value in the best solution for Case #4. (©2020 IEICE.)

Discussion on extendability

Since the extendability of the proposed method is difficult to evaluate quantitatively, this work evaluated the applicability of the proposed method under the various use cases. Although the applicability is not equal to the extendability, this work assumes that it indirectly provides evidence that the proposed method has extendability.

Figure 3.9 shows the components of each objective function value in the best solution for each case. In this evaluation, this work selected 6 use cases to discuss the effects of changing each option (Cases #1, #3, #4, #5, #8, #12) and set to each weighting parameter $\theta^{\text{link}} = \theta^{\text{server}} = \theta^{\text{r}} = 1$. Results reveal that the proposed method can improve the solution by repeating the exploration in all cases of both algorithms (w/ RL and w/o RL). Therefore, this work can

indirectly show that the proposed method has highly extendability.

Next, this work considers the effects of adding or changing each option in details. First, this work describes the result of the simplest Case #12. Figure 3.9(f) shows that the proposed method improves the maximum link utilization. Although reliability is not considered as an objective function, the total reliability is also improved. It seems that since the solution with a shorter route was preferentially selected to reduce the maximum link utilization, the total reliability was improved coordinately.

This work first considers the influence of adding the **(1) with IDS** option under the IDS sharing option by comparing Figs. 3.9(e) and 3.9(f). In Fig. 3.9(e), the maximum link utilization of the initial solution is drastically increased by adding IDS. This is because the path length increases due to the addition of the middle server node, and the total traffic volume in a physical network increases proportionally. This work also considers the influence of adding the IDS sharing option. In Fig. 3.9(e), the maximum server utilization is increased and the total reliability is decreased. The reason for increasing maximum server utilization is that large IDSs exist in IDS sharing condition. This can be seen from the fact that the maximum server utilization does not increase in Cases #1, #3, or #5. The reason the total reliability did not improve as the link utilization improved is that the IDS sharing makes it difficult to find a solution that avoids the disaster area. This difficulty to avoid the disaster area in IDS sharing condition is why the IDS that minimizes the length of the OD route without considering the reliability is preferentially selected.

Second, this work considers the influence of adding the **(2) with Reliability** option by comparing Figs. 3.9(c) and 3.9(e). The reliability is clearly improved by maintaining the link utilization efficiency and server utilization efficiency. The comparison between Figs. 3.9(a) and 3.9(d) shows a similar result.

Third, this work considers the influence of changing the **(3A) with Fixed node** option by comparing Figs. 3.9(a) and 3.9(b). Figure 3.9(a) shows that the total reliability is decreased slightly by introducing fixed clients. This is because the placement of the origin node (i.e., client node) is fixed, which makes it difficult to avoid the disaster area.

Finally, this work considers the influence of changing the **(3B) IDS isola-**

tion or sharing option by comparing Figs. 3.9(b) and 3.9(c). Figure 3.9(b) shows that the maximum server utilization is decreased and the total reliability is increased by changing the IDS isolation model. The decrease in server utilization efficiency made it easier to improve the server utilization efficiency by removing the large IDSs. The increase in total reliability made it easier to find the solutions that avoid the disaster area by IDSs isolation for each VN. The maximum link utilization in w/o RL also is increased due to the increase in the solution space by increasing the N_{ids}.

Discussion on weight parameters

Figures 3.9(c) and 3.10 show the effectiveness of weight parameters for Case #4. In this evaluation, this work sets four different conditions:

$$(\theta^{\mathrm{link}}, \theta^{\mathrm{server}}, \theta^{\mathrm{r}}) = (1,1,1), (10,0,0), (0,10,0), (0,0,10). \tag{3.18}$$

Note that the proposed method can satisfy all constraints even if each weighting parameter is 0 as shown in Fig. 3.10. The results in Fig. 3.10(a) have the best link utilization efficiency, the results in Fig. 3.10(b) have the best server utilization efficiency, and the results in Fig. 3.10(c) have the best total reliability. In particular, the maximum server utilization of 0.8 is a global optimum solution. From the above results, the proposed method can suggest a wide variety of options of solutions by adjusting the weighting parameters θ^{link}, θ^{server}, θ^{r}.

Discussion on difference to previous method

This work discusses the differences between the proposed coordinated method and previous combined approaches in terms of how easy/difficult they are to build and their problems are to solve. As described in Chapter 1, previous combined approaches need the specified algorithm to be built that simultaneously solves the combined optimization problem. This work formulated and implemented the combined optimization problem solving the VN allocation problem for Case #12, which is the simplest use case among Cases #1–12. This work also evaluates the differences in the performance of the solution between the proposed method based on the RL and the previous method based on the combined optimization problem.

This work first describes the formulation of the combined optimization problem for Case #12. The conditions and assumptions for Case #12 have already been described in Section 3.3.2. Since the control metrics are routes and VM placements, in this case, this work needs to formulate the route control algorithm and the VM control algorithm and to newly formulate the relational equations between the variables of both algorithms. This work uses the route control algorithm shown in (2)–(7) and the VM control algorithm shown in (8)–(12).

Since traffic demands between nodes $\boldsymbol{T}^{\mathrm{node}} := \{t_{pq}\}$ in (5) are determined by the traffic demands between VMs $\boldsymbol{T}^{\mathrm{vm}} := \{t_{ij}^{\mathrm{vm}}\}$ and VM placements $\boldsymbol{\Xi}^{\mathrm{vm}} := \{\xi_{ip}^{\mathrm{vm}}\}$, the relational equations between both algorithms can be formulated as follows:

$$\begin{aligned} \boldsymbol{T}^{\mathrm{node}} &= {}^{t}\boldsymbol{\Xi}^{\mathrm{vm}}\boldsymbol{T}^{\mathrm{vm}}\boldsymbol{\Xi}^{\mathrm{vm}} \\ t_{pq} &= \sum_{i\in\boldsymbol{V}}\sum_{j\in\boldsymbol{V}} \xi_{ip}^{\mathrm{vm}} t_{ij}^{\mathrm{vm}} \xi_{jq}^{\mathrm{vm}}, \end{aligned} \tag{3.19}$$

where t_{pq} shows traffic demands from node p to node q, t_{ij}^{vm} shows traffic demands from i^{th} VM to j^{th} VM, and ξ_{ip}^{vm} shows VM allocation, which returns 1 if i^{th} VM is assigned to the p^{th} node; otherwise, 0. Therefore, the combined optimization problem is formulated with the objective of minimizing $U_{\max}^{\mathrm{link}} + U_{\max}^{\mathrm{server}}$ and the constraints (2)–(12) and (18).

Next, this work describes the implementation to solve the combined optimization problem. This problem is categorized into quadratically constrained mixed-integer non-linear programming (QC-MINLP). This work uses Pyomo [68, 69], which is a Python-based open-source optimization modeling tool, and MindtPy [70], which is the Mixed-Integer Nonlinear Decomposition Toolbox in Pyomo, to solve the MINLP problem. Since the optimal solution of the MINLP problem is difficult to calculate, these tools repeat the following procedure to calculate the sub-optimal solution. These tools first decompose the MINLP problem into the continuously relaxed Non-linear Programming (NLP) problem and the Mixed-Integer Programming (MIP) problem and then calculate each problem. After calculating two problems, they consider the NLP solution as the upper bound and the MIP solution as the lower limit. They repeat the decomposition and calculation procedures until the difference between

the upper and lower bounds is sufficiently small. This work uses IPOPT [71] and MUMPS [72, 73] for solving the NLP problem and GLPK for solving the MIP problem.

As described above, the difficulty of the previous approach is that the following time-consuming tasks are required depending on the individual use cases: constructing a new formulation such as (18), selecting and combining tools to solve the combined problem such as [68–73], preparing the development environment, and implementing the combined problem. In Case #12, the number of constructing new formulation is only one because it is the simplest use case among Cases #1–12 and has only two control metrics. However, when control metrics increases, the number of new formulations needing to be constructed increases by the number of control metric combinations. On the other hand, in the proposed method, the above function can be replaced by the I/O conversion unit, which is programmable and does not need to express mathematical expressions. In addition, it enables solutions to be calculated for various use cases.

This work indicates the comparative evaluation of the performance when $N_{\mathrm{VN}} = 20$. These tools solving the MINLP problem are non-commercial and are very limited in terms of the size that can be solved. Since these tools take a long time to calculate the sub-optimal solution, this work limits the calculation time to a maximum of 10 minutes. The proposed method uses the best CEV solutions found up to 5000 steps. As shown in Fig. 3.8, the calculation time of the proposed method is 10 minutes or less. Other evaluation conditions are the same as described in Section 3.4.1.

Table 3.7 shows the average and standard deviation ($\pm\sigma$) of the performance of the solution. This work carried out 10 calculations with random initial conditions. This work first sets the same initial conditions for both methods. This work then excluded the cases with an invalid initial solution for the proposed method and when no feasible solution is found after 10 minutes calculation for the previous method. Results shows that the proposed method achieves better CEV compared to the previous method. Although the previous method outperforms when commercial tools are used, the proposed method almost matches up the previous method in terms of CEV, and is much easier to implement new control metrics.

Table 3.7: Performance of solution when Case #12 and $N_{\mathrm{VN}} = 20$.

Methods	Performance (CEV)
Propose	0.86 ± 0.043
Previous	0.63 ± 0.12

Discussion on applicability

The proposed method can be applied to the VN allocation problem that considers the combination of link resource constraints (e.g., route selection) and server resource constraints (e.g., VM placement) even if the control metrics, control objective, and network model are changed as shown in Cases #1–12. Though there are some minor constraints for the initial solution and the calculation order of each engine's evaluation, all the constraints are considered to be minor compared with the merits of the proposed method.

This work first describes the condition of the initial state. As the required condition, this work needs to find a feasible initial solution to obtain the advantage of speeding up the solution exploration by the proposed method based on RL. Because invalid solutions are concentrated near the invalid solution, an agent of RL always obtains a negative reward and cannot learn the strategy for how to find better solutions from past exploration steps.

All evaluations in this work assume that a feasible initial solution has already been found. When not finding a feasible solution, this work tries the following two ways. One is to use the proposed method against an invalid initial solution. The proposed method can explore the feasible solution through random exploration. Fortunately, once a feasible solution is found, the proposed method always converges to the better solution by RL. The other way is to use the sequential VN allocation, in which each VN demand is judged as to whether the physical network can allocate it or not when it is received. In this way, this work allocates the N^{th} VM demand to the remaining resources after the $N-1$ VNs resource optimization using the proposed method. When N^{th} VN demand cannot be allocated to the remaining resources, this request is rejected. The evaluation of the effectiveness of the two ways is for further study.

This work next describes the condition of the calculation order of each

engine's evaluation values. When calculating the CEV, the evaluation values between interdependent control metrics should be calculated simultaneously (e.g., the VM placement and IDS placement), and evaluation values between dependent control metrics should be calculated sequentially (e.g., the route between VMs is determined by the VM placements). For example, in (16), the evaluation value of VM and IDS placements (i.e., $\tilde{U}^{\text{server}}_{\text{max}}$) is calculated by aggregating the results of VM and IDS placements. The evaluation value of route (i.e., $U^{\text{link}}_{\text{max}}$) is calculated after the VM and IDS placements are determined. Similarly, the evaluation value of reliability (i.e., R^{total}_{t}) is calculated after the routes are determined.

3.5 Chapter summary

This chapter presented an extendable resource-integrated control method in network functions virtualization (NFV) by coordinating multiple control algorithms. This work developed an efficient coordination algorithm on the basis of reinforcement learning (RL), which makes it possible to find better solutions with fewer explorations by learning a strategy that can improve resource-utilization efficiency with each exploration step. Simulations revealed that the proposed algorithm can improve solution exploration for 12 representative types of the virtual network allocation use cases modeled from previous studies. This qualitatively revealed that the proposed method has extendability. This work also found that it can improve resource-utilization efficiency by 22% and total reliability by 8% in less than 5000 steps in the case of several hundred virtual machines (VMs) and a hundred intrusion detection systems (IDSs).

For future work, this work plans to evaluate the applicability of the proposed method in more complicated use cases with realistic traffic patterns and virtual network functions (VNFs) demands. This work also plans to enhance the solution-exploration-speed and scalability of the proposed coordination algorithm by using deep RL [9] and parallelization of agent learning [74].

Chapter 4

Safe multi-agent deep reinforcement learning for dynamic virtual network allocation

Network traffic and computing demand have been changing dramatically due to the growth of various types of network services, e.g., high-quality video delivery and operating system (OS) update. To maximize the utilization efficiency of limited network resources, network resource control technology is required for smooth and quick operation when the network demands change. This work proposes a dynamic virtual network allocation method based on safe multi-agent deep reinforcement learning (Safe-MADRL). A part of this work in this chapter was presented in [27]. This method can quickly optimize network resources even while network demands are drastically changing by learning the relationship between network demand patterns and optimal allocation by using the deep reinforcement learning (DRL) algorithm in advance. This work develops two techniques to be used with the proposed method; safety-considerations and multi-agent. The proposed safety-considerations technique reduces the degree of constraint violations, such as network congestion and server overload, and the proposed multi-agent technique improves the scalability of virtual network allocation by dividing demands into groups and assigning

each group's allocation to each agent. As a result of a simulation evaluation, Safe-MADRL can calculate effective allocation within one second that doubles the link utilization efficiency without any constraint violations compared to the static virtual network allocation method.

This chapter is structured as follows. Section 4.1 describes the Safe-MADRL-based dynamic VN allocation method and its modeling and formulation. Section 4.2 evaluates its performance, and Section 4.3 concludes the chapter.

4.1 Proposed method

4.1.1 Overview

With the proposed dynamic VN allocation method, an agent observes the current VN demands and physical network states, and the agent learns how to change the VN allocation to improve network resource efficiency. By applying DRL to agent learning, it can handle continuous traffic demand and high-dimensional network states. Furthermore, this work develops safety-considerations and multi-agent techniques to be used with the proposed method.

As mentioned above, this work introduces two types of agents, objective and constraint. The objective agent learns how to change the VN allocation to improve the objective function, and the constraint agent learns how to change the VN allocation to satisfy all constraints. This approach is inspired by safety exploration [41] that involves pre-training the safety layer to meet the constraints. This work also decomposes the reward function dedicated to each agent and learn a separate Q-value function for each component reward function:

$$Q(s,a) := Q_o(s,a) + w_c Q_c(s,a), \tag{4.1}$$

where Q_o and Q_c are Q-values for the objective and constraint agents. This approach is based on HRA [42]. By separating the learning to satisfy the constraints, the constraint agent does not select actions that temporarily violate the constraints for maximizing the cumulative reward. Therefore, the proposed method can improve the performance and avoid constraint violations.

Table 4.1: Symbol descriptions for dynamic VN control.

Symbols	**Definitions**
$t \in \boldsymbol{T}$	Time-step ($\boldsymbol{T}$: Total time-steps)
$\boldsymbol{N}$	Number of VNs
$\boldsymbol{G}(\boldsymbol{S}, \boldsymbol{L})$	Network graph ($\boldsymbol{S}$: server set, $\boldsymbol{L}$: link set)
$s \in \boldsymbol{S}, n \in \boldsymbol{N}$	Server, Node
link $(i, j) \in \boldsymbol{L}$	Link from server i to server j
$u^L_{ij,t}$	(i, j) link utilization at step t
$u^S_{i,t}$	i^{th} server utilization at step t
$U^L_t = \max_{ij}\left(u^L_{ij,t}\right)$	Maximum link utilization at step t
$U^S_t = \max_{i}\left(u^S_{i,t}\right)$	Maximum server utilization at step t
$\boldsymbol{D}_t := \left\{d_{i,t}\right\}$	Traffic demands of i^{th} VN at step t
$\boldsymbol{V}_t := \left\{v_{i,t}\right\}$	VM demands of i^{th} VN at step t
$\boldsymbol{R}^L_t := \left\{r^L_{ij,t}\right\}$	(i, j) residual link resources at step t
$\boldsymbol{R}^S_t := \left\{r^S_{i,t}\right\}$	i^{th} residual server resources at step t
$\boldsymbol{Y}_t := \left\{y_{ij,t}\right\}$	VM allocation at step t (VM i, server j)
$\boldsymbol{T} := \left\{t_{pq}\right\}$	Traffic matrix from server p to server q
$\boldsymbol{P} := \{p_i\}$	i^{th} User placement
c_i	Server capacity of server i
c_{ij}	Link capacity of link (i, j)

This work divides the demands of N VNs demands into groups of M agents and assigns each agent for each VN group. Note that the proposed method includes $2M$ agents, i.e., M objective agents and M constraint agents. Since DRL performance drastically decreases as the number of actions increases, as described in Section 2.1, this work aims to reduce the number of candidate actions per agent. This work also restricts the agents that could act at each time to avoid conflicts among agents.

Table 4.2: Symbol descriptions for Safe-MADRL.

Symbols	**Definitions**
M	Number of Agents
$e \in E$	Episode (E: Total episodes)
$\mathcal{G} := \{g_k\}$	Agent sets ($1 \leq k \leq 2M$)
$\mathcal{G}^o := \{g_k^o\}$	Objective agent sets ($1 \leq k \leq M$)
$\mathcal{G}^c := \{g_k^c\}$	Constraint agent sets ($1 \leq k \leq M$)
$s_t \in \mathcal{S}$	State at step t
$a_{k,t} \in \mathcal{A}_k$	Action of k^{th} agent at step t
r_t	Reward at step t ($-1 \leq r_t \leq 1$)
$Q(s_t, a_{k,t})$	Action-value function for s_t and $a_{k,t}$
$\mathcal{M}_k$	Replay memory for agent k
w_c	Weighting parameter of constraint agents $\mathcal{G}^c$

4.1.2 Modeling

Dynamic VN allocation problem

This work considers the use case of providing a cloud computing service as an example. To provide such a service, VN allocation to a physical network is needed, and each VN demand consists of a virtual node as a VM and virtual link as a route between users and VMs. This work assumes that the physical network $G(\boldsymbol{S}, \boldsymbol{L})$ consists of physical link $\boldsymbol{L}$ and physical server $\boldsymbol{S}$ and each physical server is connected to each node, i.e., $G(\boldsymbol{S}, \boldsymbol{L}) = G(\boldsymbol{N}, \boldsymbol{L})$.

Table 4.1 summarizes the definitions of the variables of the dynamic VN allocation problem. This work introduces two objective functions: minimization of total maximum link utilization U_t^L and minimization of total maximum server utilization U_t^S, i.e., min : $\sum_{t \in T} \left(U_t^L + U_t^S\right)$. This work also imposes two constraints: link capacity and server capacity, i.e., $U_t^L < 1$ and $U_t^S < 1$.

This work assumes that each user requests one VN demand and user placements are constant during the VN demand lifetime. This work considers the demands of N VNs and N users. This work also considers a discrete time-step t and assumes the VN demands change at each t. A VN demand consists of one origin (i.e., user) and one destination (i.e. VM), traffic demand $\boldsymbol{D}_t$, and

VM demand $\boldsymbol{V}_t$. The VM demand indicates the processing capacity of the VM request such as the requested number of CPU cores. When each VN demand is accepted, the amounts of link and server resources consumed depend on the traffic and VM demand. If an origin-destination (i.e., user-VM) pair is allocated in the same server, the traffic demand between the user and VM on the physical network is regarded as 0.

At the beginning of each t, the VN demands are observed. Based on the observation, the proposed method calculates the optimal VN allocation for the next $t+1$. Note that this optimal allocation is calculated based on the current observation and does not directly predict the next $t+1$ observation. Next, if necessary, controllers update the routing information and migrate each VM. Since this work focuses on the optimization of resource allocation, this work assumes the migration process is ideal, which means that the VN can be migrated without interrupting the running service.

Safe-MADRL algorithm

Table 4.2 summarizes the definitions of the variables of the proposed Safe-MADRL algorithm. This work introduces two groups of agents, i.e., a group of objective agents $\mathcal{G}^o$ and a group of constraint agents $\mathcal{G}^c$. Each group consists of $\boldsymbol{M}$ agents. That is, this work introduces $2\boldsymbol{M}$ agents:

$$\mathcal{G} = [\mathcal{G}^o, \mathcal{G}^c] = \left[g_1^o, ..., g_M^o, g_1^c, ..., g_M^c\right]. \tag{4.2}$$

The demands of N VN are divided into groups of $\boldsymbol{M}$ agents, and g_k^o and g_k^c learn how to optimize VN allocation for the k^{th} VN group. The state s_t is the same for all agents $\mathcal{G}$. The candidate action sets $\mathcal{A}_t$ of g_k^o and g_k^c are the same, and the rewards r_t for g_k^o and g_k^c differ. The learning algorithm is the same for both $\mathcal{G}^o$ and $\mathcal{G}^c$.

A state is defined as $s_t = [\boldsymbol{D}_t, \boldsymbol{V}_t, \boldsymbol{R}_t^L, \boldsymbol{R}_t^S]$, where $\boldsymbol{D}_t$ and $\boldsymbol{V}_t$ are the traffic and VM demands at t, and $\boldsymbol{R}_t^L$ and $\boldsymbol{R}_t^S$ are the residual resources of each link and server at t. Each $r_{ij,t}^L$ is calculated by $r_{ij,t}^L = 1 - c_{ij}u_{ij,t}^L$, and each $r_{i,t}^S$ is similar. The action set $\mathcal{A}_k$ is defined as a set of allocation combinations included in group k ($|\mathcal{A}_k| = |\boldsymbol{S}|^{N/M}$). Each element in $\mathcal{A}_k$ is a candidate action $a_{k,t}$ of agent g_k. As a design of rewards, this work gives a large negative value if the constraints are not satisfied; otherwise, a positive value depends

Algorithm 3 Safe-MADRL for dynamic VN allocation.

Train objective agents $\mathcal{G}^o$ by Alg. 4
Train constraint agents $\mathcal{G}^c$ by Alg. 4
for $t = 1, \infty$ **do**
    $k = t \bmod M$
    $s_t \leftarrow$ observe state
    $a_{k,t} \leftarrow \arg\max_{a' \in \mathcal{A}^k} \left[Q_o\left(s_t, a'\right) + w_c Q_c\left(s_t, a'\right)\right]$
    $s_{t+1} \leftarrow$ update environment $\left(s_t, a_{k,t}\right)$ by Alg. 5

Algorithm 4 MADRL algorithm.

make training-traffic sequences
initialize: agent parameters
for $e = 0, E$ **do**
    Set training-traffic sequences for all VNs
    initialize: environment parameters
    initialize: $s_1 \leftarrow$ observe state
    for $t = 1, T$ **do**
        $k = t \bmod M$
        $a_{k,t} \leftarrow$ select epsilon greedy action $\left(s_t\right)$
        $s_{t+1} \leftarrow$ update environment $\left(s_t, a_{k,t}\right)$ by Alg. 5
        $r_t \leftarrow$ calculate reward $\left(s_t, a_{k,t}, s_{t+1}\right)$ by Alg. 6
        store transition $\left(s_t, a_{k,t}, r_t, s_{t+1},\right)$ in $\mathcal{M}_k$
        if $r_t = -1$ **then**
            terminate episode e
        train agent by transition $(s_j, a_{k,j}, r_j, s_{j+1})$ in $\mathcal{M}_k$

on the objective function value. A certain negative value is also given when VN allocation is changed to avoid unnecessary VN reallocation. Based on the above, this work designs different rewards for $\mathcal{G}^o$ and $\mathcal{G}^c$ (detailed formulation is given in Alg. 6).

4.1.3 Formulation

Safe-MADRL algorithm

Algorithm 3 shows dynamic VN allocation using Safe-MADRL algorithm. Lines 1–2 show the pre-training of $\mathcal{G}^o$ and $\mathcal{G}^c$ by using Alg. 4 and calculation of $Q_o(s,a)$ and $Q_c(s,a)$. Next, this algorithm continually repeats lines 4–7 at every fixed t. Line 4 determine the agents to act. At each t, agents $g_k^o \in \mathcal{G}^o$ and $g_k^c \in \mathcal{G}^c$ ($k = t \bmod M$) are selected. Line 5 shows the observation of s_t from the VN environment. In line 6, an $a_{k,t}$ that maximizes $Q_o(s,a) + w_c Q_c(s,a)$ is selected, where w_c is the weighting parameter to decide the importance of avoiding constraints violation. The s_t of each agent is common in all agents, but the candidate actions of each agent differ for each k. In line 7, VN allocation is updated according to $a_{k,t}$ by Alg. 5, and returns the s_{t+1}.

MADRL algorithm

Algorithm 4 shows the MADRL algorithm for $\mathcal{G}^o$ and $\mathcal{G}^c$. The difference in the two-agent groups is only the rewards calculation.

Lines 1–2 show the generation of the training-traffic sequences and initialization of agents parameters. A series of actions is called an episode, and in each episode (lines 3–15), a series of procedures (lines 7–15) is repeatedly executed until learning is complete. In lines 9–11, learning samples that are combinations of s_t, $a_{k,t}$, r_t, and s_{t+1} are collected and stores in replay memory $\mathcal{M}_k$ in line 12. The reason for storing the samples once in replay memory is to eliminate the time dependence of collecting training samples [9]. Line 9 means the action selected on the basis of the strategy that a random action is selected with probability ε; otherwise, an action $a_{k,t}$ that maximizes $Q(s_t, a')$ is selected (i.e., $\arg\max_{a' \in \mathcal{A}^k} Q(s_t, a')$) with probability $1 - \varepsilon$. This is to avoid convergence to a local optimum solution. Line 11 shows reward calculation. Lines 13–14 means the termination condition of agent learning. In this algorithm, $r_t = -1$ is the terminate condition, i.e., the state that does not satisfy at least one constraint. In line 15, g_k is trained by a learning sample $(s_j, a_{k,j}, r_j, s_{j+1})$, which is randomly taken from $\mathcal{M}_k$.

Algorithm 5 Update environment.

1: $\boldsymbol{D}_{t+1}, \boldsymbol{V}_{t+1} \leftarrow$ update VN demand (t)
2: $\boldsymbol{Y}_{t+1} \leftarrow$ calculate VM allocation $\left(a_{k,t}, \boldsymbol{Y}_t\right)$
3: $\boldsymbol{T} \leftarrow$ convert traffic demands $(\boldsymbol{D}_{t+1}, \boldsymbol{Y}_{t+1}, \boldsymbol{P})$
4: $\boldsymbol{R}^L_{t+1}, U^L_{t+1} \leftarrow$ calculate link utilization $(\boldsymbol{T})$
5: $\boldsymbol{R}^S_{t+1}, U^S_{t+1} \leftarrow$ calculate server utilization $(\boldsymbol{V}_{\mathrm{t}}, \boldsymbol{Y}_{\mathrm{t}})$
6: **return** $s_{t+1} = [\boldsymbol{D}_{t+1}, \boldsymbol{V}_{t+1}, \boldsymbol{R}^L_{t+1}, \boldsymbol{R}^S_{t+1}]$

Update environment

In this formulation, the action sets that candidate VN allocation changes are determined only from the VM allocation, and the routes between users and VMs are uniquely determined in the environment update.

This work updates the environment based on the extendable resource-integrated control architecture [56] that coordinates multiple control algorithms specified for individual metrics. This work uses the route-optimization algorithm as the specified optimization algorithm. This formulation is described in Chapter 3. Using this architecture, the routes between users and VMs are uniquely determined when the destination server is determined.

Algorithm 5 shows the procedure of the update environment. Line 1 shows the observation of the next demands. In line 2, the next VM allocation is calculated from $a_{k,t}$ and current VM allocation. Line 3 means the calculation of the traffic matrix $\boldsymbol{T}$ from origin node, destination server, and VN traffic demands $\boldsymbol{D}_t$. The origin node is determined by the user placement $\boldsymbol{P}$, and the destination server is determined by the VM allocation $\boldsymbol{Y}_t$. Lines 4–5 shows the calculation of link and server utilization. Finally, it returns s_{t+1}.

Reward calculation

Algorithm 6 shows the procedure of the reward calculation for $\mathcal{G}^o$ and $\mathcal{G}^c$. The $\mathcal{G}^o$ learns how to maximize the objective function, and $\mathcal{G}^c$ learns how to minimize the constraint variations. Lines 1–5 shows the reward calculation of $\mathcal{G}^o$. The term $\mathrm{Eff}(x)$ in line 2 shows the efficiency function and it defined as

Algorithm 6 Reward calculation.

1: **if** $g_k \in \mathcal{G}^o$ **then**
2: $\quad r_t \leftarrow \mathrm{Eff}\left(U^L_{t+1}\right) + \mathrm{Eff}\left(U^S_{t+1}\right)$
3: $\quad$ **if** $\boldsymbol{Y}_t \neq \boldsymbol{Y}_{t+1}$ **then**
4: $\quad\quad r_t \leftarrow r_t - 0.05$
5: $\quad$ **return** $\max(-1, \min(1, r_t))$
6: **if** $g_k \in \mathcal{G}^c$ **then**
7: $\quad$ **if** $U^L_{t+1} > 1 \vee U^S_{t+1} > 1$ **then**
8: $\quad\quad$ **return** $r_t \leftarrow -1$
9: $\quad$ **else**
10: $\quad\quad$ **return** $r_t \leftarrow 0$

follows:

$$\mathrm{Eff}(x) = \begin{cases} 0.5 & (x \leq 0.2) \\ -x + 0.9 & (0.2 < x \leq 0.9) \\ -2x + 1.8 & (0.9 < x \leq 1) \\ -1.5 & (1 < x). \end{cases} \tag{4.3}$$

This function returns a positive value depending on x if $x < 0.9$; otherwise it returns a negative value. The values of 0.5 and -1.5 indicate the upper and lower limit of this function. Note that this work uses a handcrafted efficiency function in this formulation, but this function design is essentially independent of the effectiveness of the proposed method. In lines 3–4, it adds -0.05 as a penalty if the $\boldsymbol{Y}$ is changed. Finally, it returns a clipped reward within ± 1. The reason for clipping the reward is to increase the stability of agent learning [9]. Lines 6–10 shows the reward calculation of $\mathcal{G}^c$. It returns -1 if either $U^L_{t+1} > 1$ or $U^S_{t+1} > 1$; otherwise it returns 0.

4.2 Evaluation

This work evaluated the effectiveness of the proposed method through simulations in terms of performance and computation time. This work prepared three comparison methods. This work adopted Double-DQN [31] as the DRL

algorithm. This work used a three-layer fully connected DNN the input dimension of which is $|s_t|$ and output dimension is $|\mathcal{A}_k|$. This work implemented the DRL algorithm using ChainerRL [44] and route-optimization algorithm using the GNU Linear Programming Kit (GLPK) [65].

4.2.1 Evaluation conditions

For all evaluations, this work sets the number of VNs to $N = 20$, number of agents to $M = 5$, total time-steps to $T = 200$, total episodes to $E = 20000$, and weighting parameter of $\mathcal{G}^c$ to $w_c = 1$. For the VN-demand conditions, the $\boldsymbol{P}$ is randomly generated, which was fixed for all evaluations. Each $\boldsymbol{V}_t$ is randomly given an integer value within the range of 3–5, which was reset at the beginning of each episode.

The $\boldsymbol{D}_t$ is generated using the autoregressive moving average (ARMA) model within the range of 0–200 Mbps. This work generated 1000 sets of 200-step time-series traffic sequences for training and evaluation, respectively. After generated by ARMA, all the traffic data were normalized within the range of 0–200. Figure 4.1 shows the sample traffic sequences, which images real-world traffic demands including demand fluctuation. At the beginning of each episode, N time-series sequences were randomly selected from traffic data sets.

Figure 4.2 show the network topology, server capacity, and link capacity. This work used the topology of Internet2 [64], which consists of 9 nodes. This work assumed that all users and VMs are placed in all nodes. The number of candidate VM allocations was $9^{20} \simeq 1.2 \times 10^{19}$, and that of the candidate actions of g_k was $9^{20/5} = 6561$.

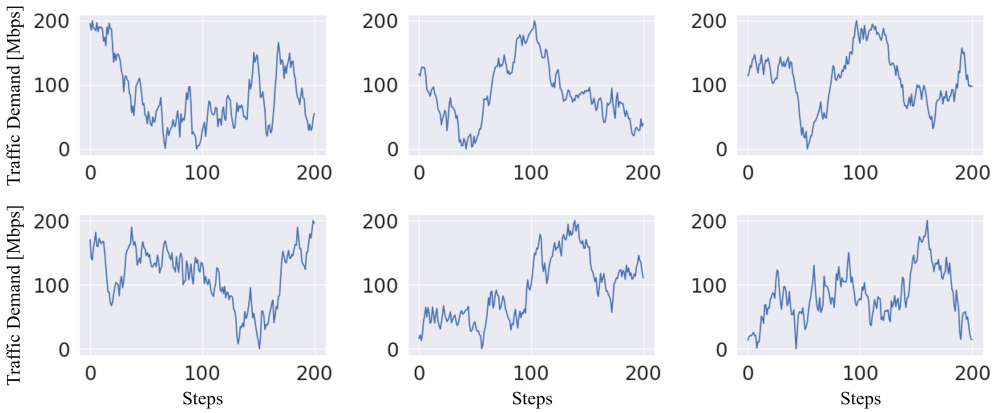


Figure 4.1: Sample traffic sequences. (©2020 IEEE.)

Figure 4.2: Network topology. (©2020 IEEE.)

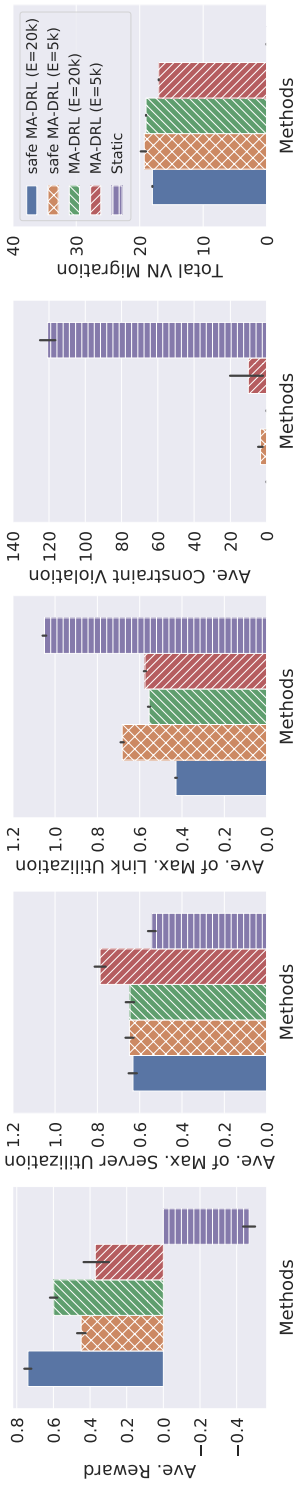


Figure 4.3: Performance evaluation for each method. (©2020 IEEE.)

4.2.2 Comparative methods

The following four comparison methods were prepared.

- **Safe-MADRL** (The proposed method with $\mathcal{G}^o$ and $\mathcal{G}^c$)
- **MADRL** (The proposed method with $\mathcal{G}^o$ and without $\mathcal{G}^c$)
- **Static** (Fixed VM allocation)
- **Exhaustive search** (Search of all candidate actions)

Note that, single-agent DRL was not evaluated because the number of actions is extremely large, and it is clear that learning is not successful.

Safe-MADRL indicates the proposed method with Alg. 3 and MADRL indicates the proposed method when only $\mathcal{G}^o$ is applied, that is, $w_c = 0$ by Alg. 3. Static is the method when VM allocation is fixed to minimize the average number of constraint violations. In this evaluation, VM allocation was fixed in the state in which the same number of VMs are allocated to all servers. Exhaustive search is the method when the action that maximizes the reward r_t is selected for all candidate actions $\mathcal{A}_k$ that agent g_k can perform in the current VN allocation. Note that all methods calculate rewards on the basis of the current demand and not on the basis of the next-step demand. Therefore, constraint violations cannot be absolutely eliminated.

4.2.3 Evaluation results

Figure 4.3 shows the average performance of each method under the practical-network conditions. This work carried out 100 calculations with random initial conditions. The width of each bar indicates the standard deviation ($\pm\sigma$). The performance of each method can roughly be compared to the average reward. The factor of performance can be considered using the 2–5$^{\text{th}}$ metrics in Fig. 4.3. That is, the performance metrics are average of maximum server utilization, average of maximum link utilization, average constraint violation, and total VM migration. When k VNs migrate at a certain time, the total VN migration adds k. Table 4.3 shows the average computation time per t of each method.

Table 4.3: Average computation time per time-step for $N = 20$.

Methods	**Time**
Safe-MADRL	102 ms
MADRL	101 ms
Static	100 ms
Exhaustive search	656 s

Performance

This work discusses the comparison between Static and the other two dynamic allocation methods. Note that Exhaustive search could not evaluate within the practical time because its computation time dramatically increased. The results show the cases for MADRL and Safe-MADRL when $E = 5000$ and 20000. Figure 4.3 shows that the two dynamic methods performed better than Static. The reason is that the dynamic allocation methods can prevent constraint violations by changing the VN allocation according to the demand change. It also indicates that the reduction in the number of actions by using the multi-agent technique is effective. The reason that MADRL can reduce the constraint violations without information of the next demand is that the RL agent indirectly predicts the next demand through learning of the optimal allocation. Safe-MADRL prevented constraint violations more than MADRL because the constraints of agent $\mathcal{G}^c$ increases the probability of taking an action that satisfies those constraints. At $E = 5000$, the constraint violation of Safe-MADRL already decreased to 1/100 that of Static. At $E = 20000$, there was no longer a violation constraint with Safe-MADRL and MADRL. Safe-MADRL also reduced the average of maximum link utilization to half that of Static while maintaining the average of maximum server utilization compared to that of Static. Therefore, the proposed method is effective for practical-network conditions.

Computation time

Table 4.3 shows the average computation time of each method, which includes the time to decide action, update the next state, and evaluate performance.

This work used Intel core i7 8700 for the evaluation. The computation time per t of all methods other than Exhaustive search was less than one second. The computation times of the three methods were almost equal because the time of the route-optimization calculation became dominant, and that of Exhaustive search drastically increased. Exhaustive search was not effective in a practical condition due to the huge computation time. Finally, this work revealed that the proposed method is also useful as a dynamic VM allocation method in terms of computation time.

4.3 Chapter summary

This chapter proposed a dynamic virtual network allocation method based on the proposed Safe-MADRL algorithm. This method can quickly optimize the network resources even when traffic demand change drastically. It can also reduce the agent's constraint violations such as the control that leads to network congestion and server overload. Simulations revealed that the proposed method can reduce the maximum link utilization to half while maintaining the maximum server utilization compared to the static allocation method under practical-network conditions. Moreover, the computation time of the proposed method was less than one second, which is a significant increase in speed compared to exhaustive search. As a result, this work revealed that the proposed method simultaneously enables efficient and immediate dynamic VN allocation. For future work, this work plans to evaluate the efficiency of the proposed method for real-world traffic demands.

Chapter 5

Cooperative multi-agent deep reinforcement learning for dynamic virtual network allocation

Network traffic and computing demand have been changing dramatically due to the growth of various types of network services, e.g., high-quality video delivery and operating system (OS) updates. To maximize the utilization efficiency of limited network resources, network resource control technology is required for smooth and quick operation when network demands change. Therefore, this work proposes a dynamic virtual network (VN) allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). A part of this work in this chapter was presented in [75, 76]. This method can quickly optimize network resources even while network demands are drastically changing by learning the relationship between network demand patterns and optimal allocation by using deep reinforcement learning (DRL) in advance. The key idea is to use a multi-agent technique for a reinforcement learning (RL) based dynamic VN allocation method, which can reduce the number of candidate actions per agent and can improve the performance for VN allocation. Moreover, a cooperation technique improves the efficiency of VN allocation. From results of a simulation evaluation, Coop-MADRL can calculate effective allocation

within one second, which reduces the maximum server and link utilization and drastically reduces the constraint violations compared with that of the static VN allocation method. Furthermore, this work revealed that the learning with various mixed traffic models could achieve a high generalization performance for all traffic patterns.

This chapter is structured as follows. Section 5.1 defines the dynamic VN allocation problem. Section 5.2 describes the Coop-MADRL-based dynamic VN allocation method and its modeling and formulation. Section 5.3 evaluates its performance, and Section 5.4 concludes the chapter.

5.1 Dynamic virtual network allocation

5.1.1 Problem definition

This work addresses the dynamic VN allocation for time-varying demand. Each VN consists of server demands as virtual nodes and traffic demands as virtual links. In this work, dynamic VN is defined as the time-varying resource requirement of virtual nodes and virtual links, and dynamic VN allocation is defined as the resource allocation for time-varying VN demands. In static allocation, each user estimates the maximum amount of demand during their lifetime in advance and pays a fixed fee based on the estimated amount. The service provider allocates the resources for the demands to maximize the VNE acceptance rate. Once accepted, the allocation is fixed until the end of the service. This provision may not be optimal when demand fluctuates. In contrast, in dynamic allocation, each user pays a minimal fee corresponding to the resources consumed at each time. The service provider dynamically reallocates the resources in accordance with the demand at each time to maximize resource utilization efficiency, which can decrease the service cost per user and minimize the negative effect of network congestion and server overload. Moreover, the dynamic allocation is made robust to sudden changes in demands by maintaining high utilization efficiency each time.

Figure 5.1 describes an overview of dynamic VN allocation. For an example of a use case, this work considers providing a cloud computing service in a physical network consisting of a wide-area network (WAN) and DCs. To

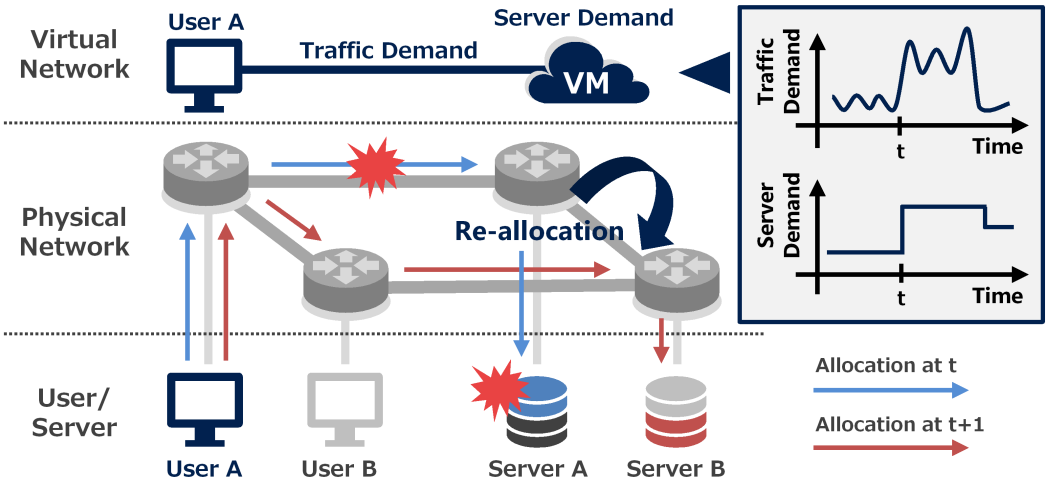


Figure 5.1: Overview of dynamic VN allocation. (©2022 IEEE.)

provide such a service, each user requests a VN demand, which consists of server demands as VMs and traffic demands between the user and VMs, and each VN demand needs to be allocated to a physical network. For simplicity, Fig. 5.1 describes a case where the number of VNs is 1, and a VN consists of one user, one VM, and the links between them. A physical network consists of 4 nodes including 2 user nodes and 2 server nodes. Here, the nodes connected to the user and server are called user nodes and server nodes, respectively. Some nodes can perform as both user and server nodes. This work considers a discrete time-step t and assumes the VN demands change at each t. As shown in the graph in Fig. 5.1, this work assumes that the two kinds of demands are time-varying, and an unexpected heavy demand suddenly emerged at time t, thereby causing the link congestion and the overload on server A. In dynamic allocation, after observing the current demands and calculating the next allocation, the VN is reallocated at $t+1$. In this example, the VM is migrated to server B, and the route is switched from a blue one to a red one.

This work focuses on immediately calculating a close-to-optimal VN allocation every t. This work uses the offline VN allocation, where all VN demands are given at the beginning of each t. This work considers K VN demands and assume that K is constant during a series of steps. When the number of VNs are changed during a series of steps, the maximum number of VNs is regarded

as K. Here, a series of steps refers to the steps until the number of VNs exceeds K. When the actual number of VNs exceeds K, the RL agent needs to be retrained. At the beginning of each t, the VN demands are observed. On the basis of the observation, the proposed method calculates the next VN allocation for the next $t+1$. Note that this next allocation is calculated on the basis of the current observation and does not directly predict the next $t+1$ observation. Since the proposed method responsively allocates the demands after observation, it must immediately calculate a close-to-optimal allocation to follow the change, e.g., within one second. This target computation time is very challenging and different from those of existing methods. Next, if necessary, controllers update the routing information and migrate each VM. After the update is complete, the proposed method proceeds to the next $t+1$.

Note that this work assumes the migration process is ideal, which means that the VM can be migrated in a short time without interrupting the running service. The way to accomplish the ideal migration is out of the scope of this work and may be considered in future research. In related work, the live migration of VM has been widely studied [77]. Moreover, the development of lightweight VMs has notably progressed. Firecracker [78], an open-source virtualization technology, enables us to deploy workloads in lightweight VMs, called microVMs. It can boot application code within 125 ms with a memory overhead of less than 5 MB per container and has been commonly used in cloud computing. This work assumes that, as VNFs evolve into cloud-native network functions (CNFs) in the future, nearly ideal migration will be made possible by utilizing these techniques.

5.1.2 Problem formulation

Table 5.1 summarizes the definitions of the physical network variables. This work assumed that the physical network graph $\boldsymbol{G(N, L)}$ consists of a physical node set $\boldsymbol{N}$ and a physical link set $\boldsymbol{L}$. Each node connected to the user and server is called user nodes and server nodes, respectively. For example, the nodes directly connected to the data center are called server nodes, and the nodes directly connected to the access network are called user nodes. Some nodes can perform as both user and server nodes. Each server node has a

Table 5.1: Symbol descriptions for physical network.

Symbols	**Definitions**
$G(\boldsymbol{N}, \boldsymbol{L})$	Network graph
$n \in \boldsymbol{N}$	Node
$s \in \boldsymbol{S}$	Server
link $(i, j) \in \boldsymbol{L}$	Link from node i to node j
c_i^S	Server capacity of server i
c_{ij}^L	Link capacity of link (i, j)
$\boldsymbol{Z} := \{z_{ij}\}$	User placement (user i, node j)

Table 5.2: Symbol descriptions for VN demands.

Symbols	**Definitions**
$t \in \boldsymbol{T}$	Time-step ($\boldsymbol{T}$: Total time steps)
$\boldsymbol{K}$	Number of VNs
$\boldsymbol{B}_t := \{b_t^i\}$	Traffic demands of i^{th} VN at step t
$\boldsymbol{D}_t := \{d_t^i\}$	VM demands of i^{th} VN at step t

Table 5.3: Symbol descriptions for control variables.

Symbols	**Definitions**
$\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$	Proportion of passed τ_t^{pq} on link (i, j)
$\boldsymbol{Y}_t := \{y_{ij,t}\}$	VM allocation at step t (VM i, server j)

server connected to it. This work denotes the server as $s \in \boldsymbol{S} \subseteq \boldsymbol{N}$. All servers and links have the capacities c_i^S and c_{ij}^L, which indicate the limit of computing resources and bandwidth resources. Each user connects to the nearest user node through the access network, which is not included in $G(\boldsymbol{N}, \boldsymbol{L})$ in this work. User placement is defined as $\boldsymbol{Z} := \{z_{ij}\}$, in which z_{ij} is 1 if the i^{th} user is connected to the j^{th} node; otherwise, 0. Here, $\boldsymbol{Z}$ is assumed to be constant during the VN demand lifetime.

Table 5.2 summarizes the definitions of the VN demand variables. A VN demand consists of one origin (i.e., user) and one destination (i.e., VM), user placement $\boldsymbol{Z}$, traffic demand $\boldsymbol{B}_t := \{b_t^i\}$, and VM demand $\boldsymbol{D}_t := \{d_t^i\}$. The i^{th} traffic demand b_t^i indicates the bandwidth between the user and VM at t, and

Table 5.4: Symbol descriptions for dynamic VN allocation.

Symbols	**Definitions**
$w \in \boldsymbol{W}$	User
$v \in \boldsymbol{V}$	VM
$\boldsymbol{\tau}_t := \left\{\tau_t^{pq}\right\}$	Traffic matrix from node p to node q
P_t	Penalty Function of VM migration
$u_{ij,t}^L$	(i, j) link utilization at step t
$u_{i,t}^S$	i^{th} server utilization at step t
$U_t^L = \max_{ij}\left(u_{ij,t}^L\right)$	Maximum link utilization at step t
$U_t^S = \max_i\left(u_{i,t}^S\right)$	Maximum server utilization at step t
$\boldsymbol{R}_t^L := \left\{r_{ij,t}^L\right\}$	(i, j) residual link resources at step t
$\boldsymbol{R}_t^S := \left\{r_{i,t}^S\right\}$	i^{th} residual server resources at step t

the i^{th} VM demand d_t^i indicates the processing power of the VM request at t such as the number of CPU cores. When each VN demand is accepted, the amount of link and server resources consumed depend on the traffic and VM demand. If an origin-destination (i.e., user-VM) pair is allocated in the same server, the traffic demand between the user and VM on the physical network is regarded as 0.

This work mainly adopted the VN demand model consisting of one origin and one destination. The problem formulation can be extended to the VN demand model with multiple origins and destinations. In this case, this work newly defines $\tilde{\boldsymbol{B}}_t := \{\tilde{b}_t^{ij}\}$ as the traffic demands between i^{th} user and j^{th} VM at step t, and defines $\tilde{\boldsymbol{D}}_t := \{\tilde{d}_t^i\}$ as the VM sizes of i^{th} VM at step t. The problem formulation and proposed method can be used directly by replacing $\boldsymbol{B}_t$ and $\boldsymbol{D}_t$ with $\tilde{\boldsymbol{B}}_t$ and $\tilde{\boldsymbol{D}}_t$.

Note that, though the VN model in this work is assumed to consist of a single VM, it can be extended to more complex VN models consisting of a graph with multiple VMs and virtual link(s) by using an extendable resource-integrated control architecture in Chapter 3 [56]. Though this formulation can be extended to more complex VN models in principle, its performance has not been evaluated yet, and its evaluation is one of the future works.

For the above physical network graph and VN demand, this work formulates the dynamic VN allocation problem. Table 5.3 summarizes the definitions of the control variables. The goal of this problem is to find an optimal VN allocation consisting of $\boldsymbol{X}_t$ and $\boldsymbol{Y}_t$ every t. Here, $\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$ shows the proportion of traffic τ_t^{pq} from origin node p to destination node q passing through link (i, j), and $\boldsymbol{Y}_t := \left\{y_{ij,t}\right\}$ shows the VM allocation in which $y_{ij,t}$ is 1 if the i^{th} VM is assigned to the j^{th} server; otherwise, 0.

Table 5.4 summarizes the definitions of the variables of the dynamic VN allocation problem. This work introduces an objective function:

$$\min : \sum_{t \in T} \left(U_t^S + U_t^L + \alpha P_t \right), \tag{5.1}$$

where U_t^S and U_t^L show the maximum server utilization and maximum link utilization at t, and P_t shows the penalty of VM migration. Here, α is a positive value and determines the degree of VM migration. This work sets the penalty to depend on the number of migrated VMs, which is formulated as follows:

$$P_t = \frac{1}{2} \sum_{i \in V} \sum_{j \in S} \| y_{ij,t} - y_{ij,t-1} \|. \tag{5.2}$$

Equation (5.2) calculates the sum of the absolute value of the difference between current $\boldsymbol{Y}_t$ and previous $\boldsymbol{Y}_{t-1}$. Since $y_{ij,t}$ and $y_{ij,t-1}$ are binary numbers, the sum of the absolute values of these differences indicates the number of migrated VMs at t. This work also imposes two constraints: link capacity and server capacity, i.e., $s.t. : U_t^S < 1$ and $U_t^L < 1$. To formulate this, a VM allocation variable $y_{ij,t}$ is formulated to minimize the server utilization U_t^S while satisfying the constraints as follows:

$$\text{s.t.} \quad : \quad \sum_{j \in \boldsymbol{S}} y_{ij,t} = 1 \qquad (\forall i \in \boldsymbol{V}) \tag{5.3}$$

$$\sum_{i \in \boldsymbol{V}} d_t^i y_{ij,t} \leq c_j^S U_t^S \qquad (\forall j \in \boldsymbol{S}) \tag{5.4}$$

$$y_{ij,t} \in \{0, 1\} \tag{5.5}$$

$$0 \leq U_t^S \leq 1. \tag{5.6}$$

Equation (5.3) shows the VM conservation law. In other words, it shows that each VM must be allocated to any server. Equation (5.4) shows the constraint

of server capacity, and Eqs. (5.5)–(5.6) show the range of variables. In addition, a routing variable $x_{ij,t}^{pq}$ is formulated to minimize the link utilization U_t^L while satisfying the constraints as follows:

$$\text{s.t.} \quad : \quad \sum_{j:(i,j)\in \boldsymbol{L}} x_{ij,t}^{pq} - \sum_{j:(j,i)\in \boldsymbol{L}} x_{ji,t}^{pq} = 0 \tag{5.7}$$

$$(\forall p, q \in \boldsymbol{N}, i \neq p, i \neq q)$$

$$\sum_{j:(i,j)\in \boldsymbol{L}} x_{ij,t}^{pq} - \sum_{j:(j,i)\in \boldsymbol{L}} x_{ji,t}^{pq} = 1 \tag{5.8}$$

$$(\forall p, q \in \boldsymbol{N}, i = p)$$

$$\sum_{p,q\in \boldsymbol{N}} \tau_t^{pq} x_{ij,t}^{pq} \leq c_{ij}^L U_t^L \tag{5.9}$$

$$(\forall (i,j) \in \boldsymbol{L}, \forall p, q \in \boldsymbol{N})$$

$$0 \leq x_{ij,t}^{pq} \leq 1 \quad (\forall (i,j) \in \boldsymbol{L}, \forall p, q \in \boldsymbol{N}) \tag{5.10}$$

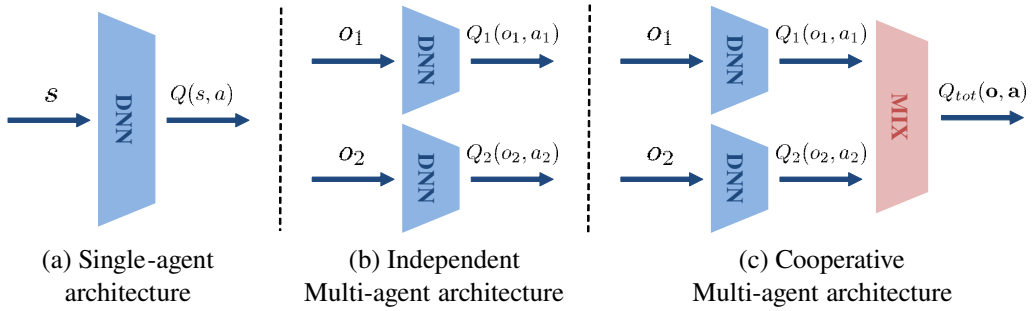
$$0 \leq U_t^L \leq 1. \tag{5.11}$$

Equations (5.7)–(5.8) show the traffic flow conservation law. Equation (5.7) shows that the traffic flowing into a node equals the traffic flowing out of the node except the source node p and destination node q. Equation (5.8) shows that the net flow out of the source node p is 1. The traffic flow conservation law at the destination node q is guaranteed when Eqs. (5.7)–(5.8) are satisfied, which is proved in [79]. Equation (5.9) shows the constraint of link capacity, and Eqs. (5.10)–(5.11) show the range of variables. Since traffic demands between nodes τ_t^{pq} in Eq. (5.9) are determined by the traffic demands $\boldsymbol{B}_t$, user placements $\boldsymbol{Z}$, and VM allocation $\boldsymbol{Y}_t$, the relational equations between both constraints can be formulated as follows:

$$\begin{aligned} \boldsymbol{\tau}_t &= \boldsymbol{Z}^\top \boldsymbol{B}_t^\top \boldsymbol{Y}_t \\ \tau_t^{pq} &= \sum_{i\in \boldsymbol{W}} \sum_{j\in \boldsymbol{V}} z_{ip} b_t^i y_{jq,t}. \end{aligned} \tag{5.12}$$

Here, this work defines a traffic matrix as $\boldsymbol{\tau}_t := \left\{\tau_t^{pq}\right\}$ and each τ_t^{pq} is a positive real number. Equation (5.12) converts the VN traffic demands into the traffic matrix of the physical network.

The dynamic VN allocation problem is formulated to minimize Eq. (5.1) and the constraints Eqs. (5.3)–(5.12). This problem is NP-hard [66] and categorized mixed-integer nonlinear problems. Since the optimal solution takes a



(a) Single-agent architecture (b) Independent Multi-agent architecture (c) Cooperative Multi-agent architecture

Figure 5.2: Three types of DRL architectures: (a) single-agent, (b) independent multi-agent, and (c) cooperative multi-agent. (©2022 IEEE.)

long time to calculate, most previous studies have targeted small-scale or static VN demands. Conversely, this work uses RL to solve the problem. Whereas the methods not based on RL take a long time to find the optimal solution each time, RL can instantly output the close-to-optimal solution by learning the relationship between resource demand patterns and optimal VN allocation in advance. However, RL needs to efficiently learn the optimal allocation for a wide variety of demand patterns to improve performance. Therefore, this work develops a cooperative multi-agent technique for an RL-based method to prevent VN allocation decisions from exponentially increasing.

5.2 Proposed method

5.2.1 Overview

In the RL-based dynamic VN allocation method, an agent observes the current VN demands and physical network states and learns how to change the VN allocation to more efficiently use network resources. By applying DRL to agent learning, the agent can handle continuous traffic demand and high-dimensional network states. Furthermore, to reduce the action space, this work develops a cooperative multi-agent technique to be used with the proposed method.

Figure 5.2 shows three types of DRL architectures: single-agent, independent multi-agent, and cooperative multi-agent. To simplify the discussion, this work assumes the number of agents is set to 2 in this figure. In the single-agent

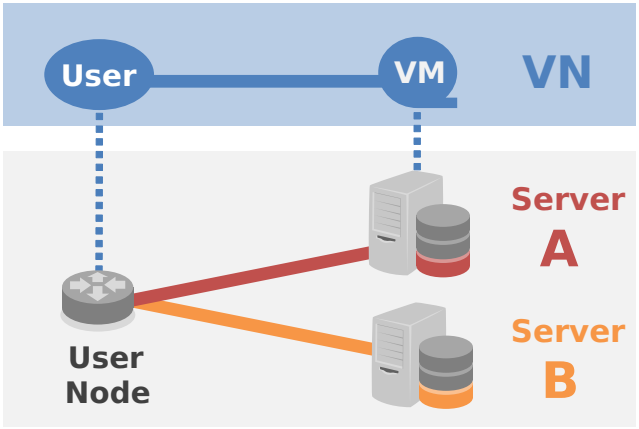


Figure 5.3: Overview of VN allocation in a simple network topology. (©2022 IEEE.)

case, an agent outputs $Q(s, a)$ with the global state s as input, where state s includes all information of a physical network and VNs. The dimension of output is equal to the action space $|\mathcal{A}|$ and is determined by the combination of all VN allocations. When assuming there are $|\boldsymbol{S}|$ servers, K VN demands, and the shortest path from the origin to destination is chosen, the number of candidate actions $|\mathcal{A}| = |\boldsymbol{S}|^K$. Moreover, as $|\boldsymbol{S}|$ and K increase, $|\mathcal{A}|$ increases exponentially. In the multi-agent case, each VN allocation control is assigned to each agent. The k^{th} agent outputs $Q_k(o^k, a^k)$ with its own local observation o^k as input, where observation o^k is part of the global state s and is related to a physical network and k^{th} VN information. The dimension of output is equal to the action space $\left|\mathcal{A}^k\right|$, which is determined only by k^{th} VN allocation, i.e., $\left|\mathcal{A}^k\right|$ drastically decreases to $|S|$.

This work describes the multi-agent behavior in a simple example. Figure 5.3 shows the VN allocation for a simple network topology consisting of only two servers. Since the path is uniquely determined, each VN allocation is determined from two choices; the VM is assigned to server A or server B. In the single-agent case, if $K = 4$, the agent should find the optimal allocation from $|\mathcal{A}| = 2^4$ actions. However, if $K = 10$, the agent should find the optimal allocation from 2^{10} actions. The multi-agent technique reduces the number of candidate actions per agent. This work prepared K agents for K VN demands,

and each agent decides each VN allocation. In the multi-agent case, whether $K = 4$ or $K = 10$, k^{th} agent only finds the optimal allocation of k^{th} VN from the $\left|\mathcal{A}^k\right| = 2$ actions. In other words, each agent only decides whether it is better to assign each VM to server A or server B.

When each agent is independent, the non-stationary learning problem described in Chapter 1 arises. In the cooperative multi-agent case, a mixed layer that calculates a joint action-value function Q_{tot} from each action-value function Q_k is added to cooperate with each agent. In the training phase, the loss of DNN is calculated by the shared reward and the joint action value Q_{tot}, thereby solving the non-stationary learning problem between agents and improving performance. It corresponds to *centralized training*. In the actual control phase, each agent can determine the best action on the basis of each observation o^k and each action value Q_k because the forward network of each agent DNN learns to output Q_k to minimize Q_{tot} during the training phase. In other words, agents can output globally optimal action by only calculating the forward network of their DNNs without a mixed network calculation. It corresponds to *decentralized execution* without other agent information.

5.2.2 Modeling

Table 5.5 summarizes the definitions of the variables of Coop-MADRL. This work introduces K agents equal to the number of VNs. Each VN control is assigned to each agent, and the k^{th} agent learns how to optimize VN allocation for the k^{th} VN.

A state is defined as $s_t = [\boldsymbol{B}_t, \boldsymbol{D}_t, \boldsymbol{R}_t^L, \boldsymbol{R}_t^S]$, where $\boldsymbol{B}_t$ and $\boldsymbol{D}_t$ are the traffic and VM demands at t, and $\boldsymbol{R}_t^L$ and $\boldsymbol{R}_t^S$ are the residual resources of each link and server at t. Each $r_{ij,t}^L$ is calculated by $r_{ij,t}^L = 1 - c_{ij}^L u_{ij,t}^L$, and each $r_{i,t}^S$ is similar. An observation for agent g_k is defined as $o_t^k = [b_t^k, d_t^k, \boldsymbol{R}_t^L, \boldsymbol{R}_t^S]$. The state represents fully observed global information, and the observation represents agent-dependent information. Here, each VN demand such as b_t^k and d_t^k shows local information, and a physical resource such as $\boldsymbol{R}_t^L$ and $\boldsymbol{R}_t^S$ shows global information. By including the residual resources in agents' observation, each agent can take into account not only its demand information but also network-wide information. The action set $\mathcal{A}^k$ is defined as a set of allocation

Table 5.5: Symbol descriptions for Coop-MADRL.

Symbols	**Definitions**
$e \in E$	Episode (E: Total episodes)
$\mathcal{G} := \{g_k\}$	Agent set ($1 \leq k \leq K$)
$s_t \in \mathcal{S}$	State at step t ($\mathcal{S}$: State space)
$\mathcal{O} := \{\mathcal{O}^k\}$	Observation sets for all agents
$o_t^k \in \mathcal{O}^k$	Observation for agent g_k at step t
$\boldsymbol{o}_t := \{o_t^k\}$	All observation at step t
$\mathcal{A} := \{\mathcal{A}^k\}$	Action sets for all agents ($\mathcal{A}^k$: Action space)
$a_t^k \in \mathcal{A}^k$	Action for agent g_k at step t
$\boldsymbol{a}_t := \{a_t^k\}$	All action at step t
r_t	Reward for agent g_k at step t
$Q_k\left(o_t^k, a_t^k\right)$	Action-value function for agent k
$Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$	Joint action-value function for all agent
$\mathcal{M}$	Replay memory
$h^i \in \boldsymbol{h}$	observation-action history
$\boldsymbol{h}$	observation-action history of all agents

combinations included in agent g_k ($\left|\mathcal{A}^k\right| = |\boldsymbol{S}|$). To design rewards, this work gives a large negative value if the constraints are not satisfied; otherwise, a positive value depends on the objective function value. A certain negative value is also given when VN allocation is changed to avoid unnecessary VN reallocation.

5.2.3 Formulation

The Coop-MADRL-based dynamic VN allocation method consists of two phases: centralized training and decentralized execution. The decentralized agents continually execute the dynamic VN allocation control after centralized training.

Algorithm 7 shows the centralized training of Coop-MADRL. Line 1 shows the initialization of agent parameters. A series of procedures (lines 2–15) is repeatedly executed until learning is complete. Lines 3–4 show the generation of the training-traffic sequences and the initialization of environment parameters and observation. A series of actions is called an episode, and each episode

Algorithm 7 Centralized training of Coop-MADRL.

1: **initialize:** agent parameters
2: **while** $t < T$ **do**
3: generate training-traffic sequences for all VNs
4: **initialize:** environment parameters
5: **for** $e = 0, E$ **do**
6: **for each** $g_k \in \mathcal{G}$ **do**
7: $o_t^k \leftarrow$ observation
8: $a_t^k \leftarrow$ select epsilon greedy action $\left(o_t^k\right)$
9: $\boldsymbol{o}_{t+1} \leftarrow$ update environment $(\boldsymbol{o}_t, \boldsymbol{a}_t)$ by Alg. 9
10: $r_t \leftarrow$ calculate reward $(\boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{o}_{t+1})$ by Alg. 10
11: **if** $r_t \leq -1$ **then**
12: terminate episode: $e \leftarrow E$
13: $t \leftarrow t + 1$
14: store episodic transition $\left(\boldsymbol{o}_j, \boldsymbol{a}_j, r_j\right), \forall j \in$ episode steps
15: train all agents $\mathcal{G}$ by random episodic transition

Algorithm 8 Dynamic VN allocation using Coop-MADRL.

1: $Q_k\left(o^k, a^k\right) \leftarrow$ train all agents $\mathcal{G}$ by Alg. 7
2: **while** True **do**
3: **for each** $g_k \in \mathcal{G}$ **do**
4: $o_t^k \leftarrow$ observation
5: $a_t^k \leftarrow \arg\max_{a' \in \mathcal{A}^k} Q_k\left(o_t^k, a'\right)$
6: $\boldsymbol{o}_{t+1} \leftarrow$ update environment $(\boldsymbol{o}_t, \boldsymbol{a}_t)$ by Alg. 9
7: $t \leftarrow t + 1$

(lines 5–13) is repeatedly executed. In each episode, agents collect learning samples that are combinations of $\langle \boldsymbol{o}_t, \boldsymbol{a}_t, r_t, \boldsymbol{o}_{t+1} \rangle$. Each agent executes lines 6–8 in parallel. Line 7 shows the observation of o_t^k from the VN environment. Line 8 means the action selected on the basis of the strategy that a random action is selected with probability ε; otherwise, an action a_t^k that maximizes $Q_k\left(o_t^k, a'\right)$ is selected (i.e., $\arg\max_{a' \in \mathcal{A}^k} Q_k\left(o_t^k, a'\right)$) with probability $1-\varepsilon$. This is to avoid convergence to a local optimum solution. In line 9, VN allocation is

Algorithm 9 Update environment.

1: $\boldsymbol{D}_{t+1}, \boldsymbol{B}_{t+1} \leftarrow$ update VN demand (t)
2: $\boldsymbol{Y}_{t+1} \leftarrow$ calculate next VM allocation $(\boldsymbol{a}_t, \boldsymbol{Y}_t)$
3: $\boldsymbol{\tau}_t \leftarrow$ convert traffic demands $(\boldsymbol{D}_{t+1}, \boldsymbol{Y}_{t+1}, \boldsymbol{Z})$
4: $\boldsymbol{R}^L_{t+1}, U^L_{t+1} \leftarrow$ calculate next link utilization $(\boldsymbol{\tau}_{\mathrm{t}})$
5: $\boldsymbol{R}^S_{t+1}, U^S_{t+1} \leftarrow$ calculate next server utilization $(\boldsymbol{D}_{\mathrm{t}}, \boldsymbol{Y}_{\mathrm{t}})$
6: $o^k_{t+1} = [d^k_{t+1}, b^k_{t+1}, \boldsymbol{R}^L_{t+1}, \boldsymbol{R}^S_{t+1}], \forall k$
7: **return** $\boldsymbol{o}_{t+1} = [o^1_{t+1}, \ldots, o^K_{t+1}]$

updated in accordance with $\boldsymbol{a}_t$ by Alg. 9 and returns the $\boldsymbol{o}_{t+1}$. Line 10 shows the reward calculation. Lines 11–12 mean the termination condition of agent learning. In this algorithm, $r_t \leq -1$ is the termination condition, i.e., the state that does not satisfy at least one constraint. Line 14 shows stores in replay memory $\mathcal{M}$. The reason for storing the samples once in replay memory is to eliminate the time dependence of collecting training samples [9]. In line 15, all agents $\mathcal{G}$ are trained by the history of episodic transition, which is randomly taken from $\mathcal{M}$.

This work describes the DNN architecture of Coop-MADRL. As shown in Fig. 5.2, DNN architecture consists of each agent's DNN layer and a mixed layer. For each agent's DNN layer, this work introduced DRQN [45] to handle time-series data as input, which incorporates recurrent neural networks (RNNs) into DQN. This work used a three-layer NN consisting of two fully connected layers and the gated recurrent unit (GRU) layer [80]. This work adopted Double-DQN [31] as the DRL algorithm. For the mixed layer, this work uses a mixed layer of VDN and QMIX as it is.

Algorithm 8 shows dynamic VN allocation using the Coop-MADRL. Lines 1 show the pre-training of $\mathcal{G}$ by using Alg. 7 and calculation of $Q_k\left(o^k, a^k\right)$. Next, this algorithm continually repeats lines 2–7 at every fixed t. In line 5, each agent selects an a^k_t that maximizes $Q_k\left(o^k, a^k\right)$.

5.2.4 Update environment

In this formulation, the action sets that are changed by candidate VN allocation are determined only from the VM allocation, and the routes between

Algorithm 10 Reward calculation.

1: $r_t \leftarrow \mathrm{Eff}\left(U^L_{t+1}\right) + \mathrm{Eff}\left(U^S_{t+1}\right)$
2: **if** $\boldsymbol{Y}_t \neq \boldsymbol{Y}_{t+1}$ **then**
3: $\quad r_t \leftarrow r_t - \alpha P_t$
4: **return** $\max(-5, \min(1, r_t))$

users and VMs are uniquely determined in the environment update. This work updates the environment on the basis of the extendable resource-integrated control architecture [56] that coordinates multiple control algorithms specified for individual metrics. This work uses the route-optimization algorithm as the specified optimization algorithm. By using this architecture, the routes between users and VMs are uniquely determined when the destination server is determined. The route-optimization calculates a routing variable $x^{pq}_{ij,t}$ to minimize the link utilization U^L_t while satisfying the constraints in Eqs. (5.7)–(5.11). Since the routing variable $x^{pq}_{ij,t}$ is a continuous value within 0–1 as shown in Eq. (5.11), this problem class is classified as a LP problem. If the routing variable $x^{pq}_{ij,t}$ is changed to a binary variable, it is classified as an MIP problem.

Algorithm 9 shows the procedure of the update environment. Line 1 shows the observation of the next demands. In line 2, the next VM allocation is calculated from $\boldsymbol{a}_t$ and current VM allocation. Line 3 means the calculation of the traffic matrix $\boldsymbol{\tau}_t$ from the origin node, destination server, and VN traffic demands $\boldsymbol{B}_t$. The origin node is determined by the user placement $\boldsymbol{Z}$, and the destination server is determined by the VM allocation $\boldsymbol{Y}_t$. Lines 4–5 show the calculation of next link and server utilization. Finally, Algorithm 9 returns $\boldsymbol{o}_{t+1}$.

5.2.5 Reward calculation

This work designs the reward function on the basis of the objective function Eq. (5.1). Algorithm 10 shows the procedure of the reward calculation for $\mathcal{G}$. The $\mathcal{G}$ learns how to maximize the reward. The term $\mathrm{Eff}(x)$ in Alg. 10 shows

the efficiency function and is defined as follows:

$$\mathrm{Eff}(x) = \begin{cases} 0.5 & (x \leq 0.4) \\ -x + 0.9 & (0.4 < x \leq 0.9) \\ -2x + 1.8 & (0.9 < x \leq 1) \\ -x - 0.5 & (1 < x). \end{cases} \tag{5.13}$$

This function returns a positive value depending on x if $x < 0.9$; otherwise it returns a negative value. This work designed a handcrafted efficiency function so that the efficiency decreases as x increases. The decrease of efficiency doubles when x is more than 0.9, and the values of 0.5 indicate the upper limit of this function. This work also designed the function to drastically decrease efficiency when $x > 1$, i.e., when the constraints are not satisfied. Note that this design is just one example that the proposed method performed suitably, and there is still room for improvement in the design of the efficient function. Elucidating the relationship between reward function and VN allocation performance remains one of the future challenges.

Line 1 show the $\mathrm{Eff}(x)$ calculation. In lines 2–3, it adds $-\alpha \boldsymbol{P}_t$ as a penalty shown in Eq. (5.2) if the $\boldsymbol{Y}_t$ is changed. Finally, it returns a clipped reward within $-5 \leq r_t \leq 1$. The reason for clipping the reward is to increase the stability of agent learning [9].

5.3 Evaluation

This work evaluated the effectiveness of the proposed method through simulations in terms of performance, computation time, and scalability for the number of VNs and network topology size. This work also evaluated the generalization performance for unknown traffic patterns. This work prepared two environments and five comparison methods. This work implemented the DRL algorithm based PyTorch [81] and PyMARL [82] and route-optimization algorithm using the GNU Linear Programming Kit (GLPK) [65].

5.3.1 Evaluation conditions

This work sets the number of VNs to $K = 20$ as the default settings. In evaluating scalability, this work increased the K from 20 to 60 and increased

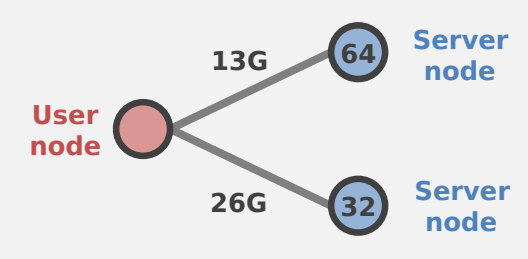


Figure 5.4: Simple network topology. (©2022 IEEE.)

link capacity and server capacity in proportion to K. This work also sets total time steps to $T = 3.0 \times 10^5$ and total episodes to $E = 200$, and the weight of the penalty function of VM migration to $\alpha = 0.01$ for all evaluations.

For the VN-demand conditions, the $\boldsymbol{Z}$ is randomly generated and fixed for all evaluations. The $\boldsymbol{D}_t$ is randomly generated integer values within 1–5 and is reset at the beginning of each episode. The $\boldsymbol{B}_t$ is 200-step time-series sequences randomly generated by various traffic models at the beginning of each episode. After generating traffic sequences, all the traffic data were normalized so that the average traffic volume was 1 Gbps and the minimum traffic volume was 0. This work generated the $\boldsymbol{B}_t$ for training and evaluation. The details of each model are described in Section 5.3.2.

For the physical-network conditions, this work prepared simple and practical networks. Figures 5.4 and 5.5 show the network topology for the simple and practical networks, respectively. This work assumed that all users and VMs are placed in the user node and server node, respectively. The values in these figures show the server capacity and link capacity when $K = 20$. When K increases, server capacity and link capacity increase in proportion to K. All evaluations except for the scalability evaluation for the network topology size are on the basis of these two topologies. This work prepared other topologies to evaluate scalability for the topology size (see Section 5.3.4 in detail).

Simple-network conditions

This work used the 3-node topology, which consists of 1 user node and 2 server nodes. When $K = 20$, the number of candidate VM allocations was $2^{20} \simeq 1.0 \times 10^6$, and that of the candidate actions of g_k was 2.

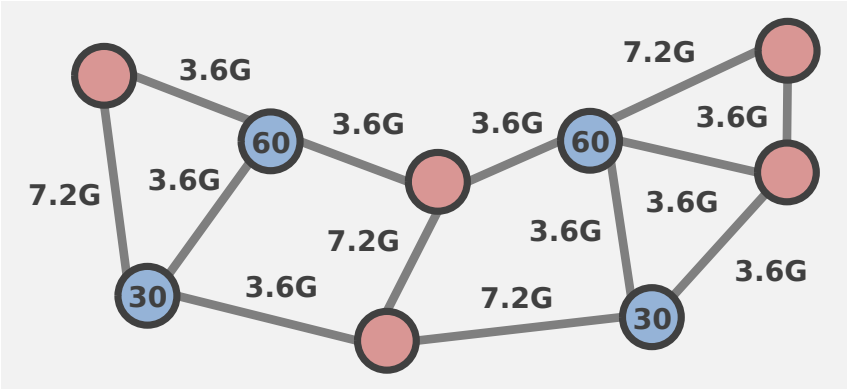


Figure 5.5: Practical network topology. (©2022 IEEE.)

Practical-network conditions

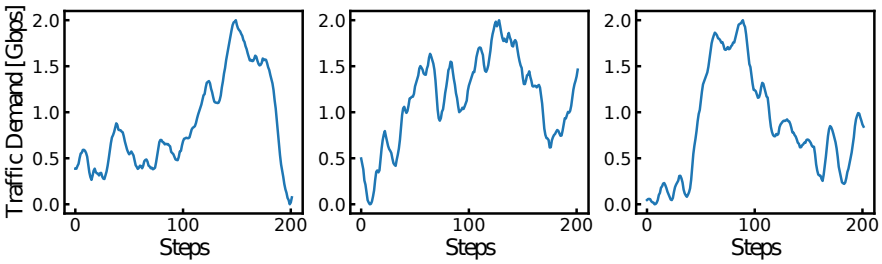
This work used the 9-node topology based on Internet2 [64], which consists of 5 user nodes and 4 server nodes as shown in Fig. 5.5, and all users and all VMs are allocated in user nodes and server nodes, respectively. When $K = 20$, the number of candidate VM allocations was $4^{20} \simeq 1.1 \times 10^{12}$, and that of the candidate actions of g_k was 4.

5.3.2 Traffic models

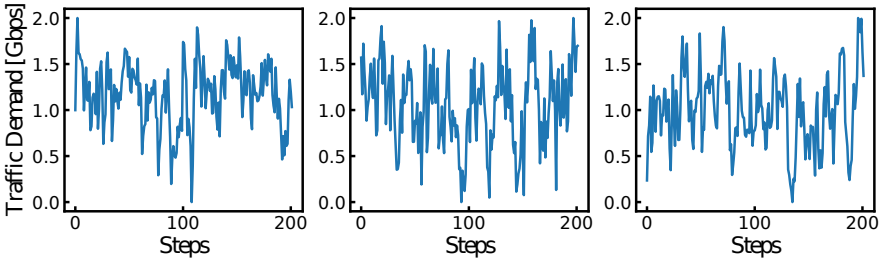
Figures 5.6 and 5.7 show the six types of generated traffic sequences used in this evaluation. It shows the generated samples of traffic sequences normalized within 0–2 Gbps after generating them. To evaluate the performance of the proposed method under various traffic patterns, this work prepared five different models to generate time-series traffic sequences and one option to modify the generated traffic sequences. This option can apply to all traffic models, which assumes the case when unexpected heavy traffic demand suddenly emerges. This work describes the models and option as follows.

ARMA model

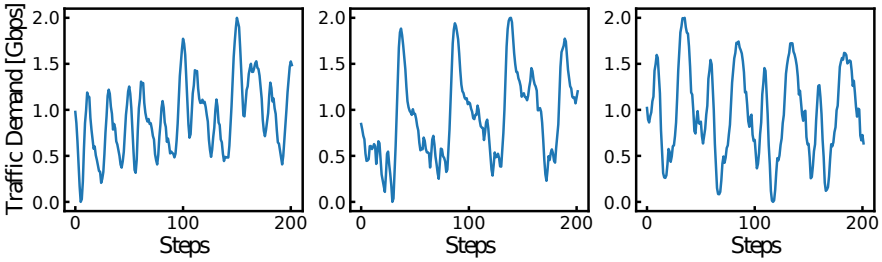
Figure 5.6(a) shows the traffic sequences generated by the autoregressive moving average (ARMA) model. This model assumes general traffic patterns, such as aggregated broadband traffic. The (p, q)–order ARMA model is formulated



(a) ARMA model.



(b) ARMA model with other parameters.



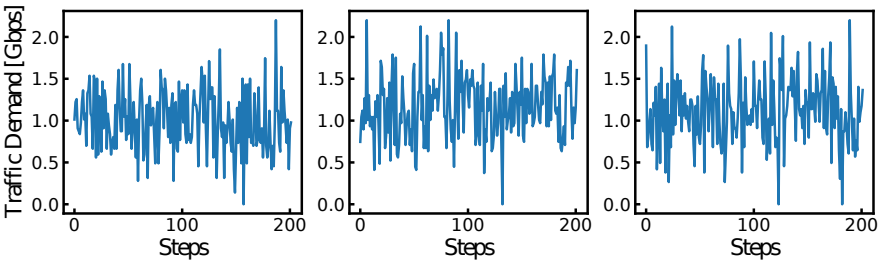
(c) SARIMA model.

Figure 5.6: Various traffic models (1). (©2022 IEEE.)

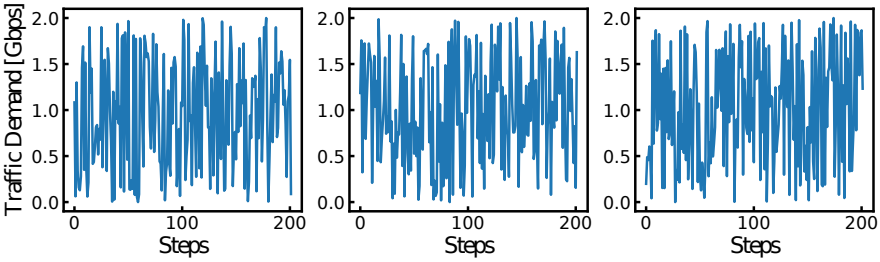
as follows:

$$f(t) = \varepsilon_t + \sum_{i=1}^{p} \phi_i f(t-i) + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}, \tag{5.14}$$

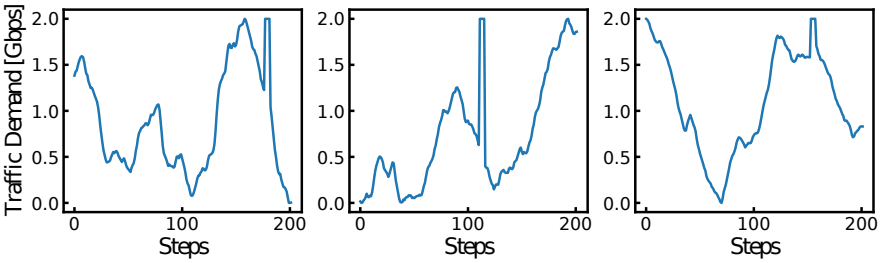
where $f(t)$ is the time-series data at time-step t, p is the order of AR(p), q is the order of MA(q), ϕ_i are AR(p) model parameters, θ_i are MA(q) model parameters, and ε_t is the distributed error at time-step t. This work sets $p = 2$, $q = 50$, and $\varepsilon_t \sim \mathcal{N}(0, 1)$. Here, $\mathcal{N}(0, 1)$ is the normal distribution with a mean of 0 and variance of 1. This work also sets hyper-parameters $\phi_1 = 0.9$, $\phi_2 = -0.1$, and $\theta_i = 0.95$ ($\forall i, 1 \leq i \leq 50$). These hyper-parameters were set to



(a) Poisson model.



(b) Random model.



(c) ARMA model with anomaly option.

Figure 5.7: Various traffic models (2). (©2022 IEEE.)

represent long-term fluctuations.

ARMA model with other parameters

Figure 5.6(b) shows the traffic sequences generated by the ARMA model when changing the hyper-parameters from Fig. 5.6(a). This work sets $p = 2$, $q = 5$ for Eq. (5.14) in this model. This work also sets hyper-parameters $\phi_1 = 0.5$, $\phi_2 = -0.1$, and $\theta_i = 0.15$ $(\forall i, 1 \leq i \leq 5)$. These hyper-parameters were set to represent short-term fluctuations. This work changed q and θ_i from (1) ARMA model, which is equivalent to weakening the effect of MA factors.

SARIMA model

Figure 5.6(c) shows the traffic sequences generated by the seasonal autoregressive integrated moving average (Seasonal ARIMA; SARIMA) model. This model assumes periodical traffic patterns, such as daily traffic trends. The $(p, d, q) \times (P, D, Q)_m$–order SARIMA model is formulated as follows:

$$\phi_p(B)\Phi_P(B)(1-B)^d(1-B^m)^D f(t) = \theta_q(B)\Theta_Q(B)\varepsilon_t, \tag{5.15}$$

where $f(t)$ is the time-series data at time-step t, B is the backward shift operator and is defined as $B^k f(t) := f(t-k)$. The $\phi_p(B)$ and $\Phi_P(B)$ are called AR operators, and the $\theta_q(B)$ and $\Theta_Q(B)$ are called MA operators. Each operator is defined as follows:

$$\phi_p(B) := 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \tag{5.16}$$

$$\theta_q(B) := 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q \tag{5.17}$$

$$\Phi_P(B) := 1 - \Phi_1 B^m - \Phi_2 B^{2m} - \cdots - \Phi_P B^{Pm} \tag{5.18}$$

$$\Theta_Q(B) := 1 - \Theta_1 B^m - \Theta_2 B^{2m} - \cdots - \Theta_Q B^{Qm}. \tag{5.19}$$

Here, (p, d, q) and (P, D, Q) show the order of the ARIMA model, and m shows the period of time-serial sequences. This work sets $(p, d, q) = (2, 0, 5)$, $(P, D, Q) = (1, 0, 1)$, and $m = 50$. This work also sets hyper-parameters $\phi_1 = 0.9$, $\phi_2 = -0.1$, $\theta_i = 1 - 0.1 \times i$ $(\forall i, 1 \leq i \leq 5)$ and $\Phi_1 = \Theta_1 = 0.9$. These hyper-parameters were set to represent periodical fluctuations based on (2) ARMA model with other parameters.

Poisson model

Figure 5.7(a) shows the traffic sequences generated by the Poisson process. This model assumes traffic patterns for IoT applications. The traffic model for IoT applications is often described by periodic patterns from asynchronous sources. This superposition of IoT traffic streams can be approximated by the Poisson process [83]. The Poisson model is formulated as follows:

$$f(t) = v\mathcal{P}_t,\ \mathcal{P}_t \sim \Pr(X{=}k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

where $\mathcal{P}_t$ is the Poisson distribution, the parameter λ is the number of IoT devices per unit time, and the v is the average traffic volume per IoT device.

Table 5.6: Summary of the proposed methods and comparison methods.

Methods	**Methodology**	**Dynamic**	**Cooperation**
QMIX (Ours)	Coop-MADRL	✓	✓
VDN (Ours)	Coop-MADRL	✓	✓
IQL	MADRL	✓	-
Static Allocation (SA)	Heuristic	-	-
Exhaustive Search (ES)	Meta-Heuristic	✓	-

Random model

Figure 5.7(b) shows the traffic sequences generated by white noise. This work sets the traffic volume at each time-step to a random value in the range of 0–1.

With anomaly option

Figure 5.7(c) shows the traffic sequences with the anomaly option applied to the ARMA model. This work randomly injected 5 steps between 0–200 steps as anomaly steps. This work sets the traffic volume of an anomaly step to 2 Gbps on the assumption that unexpected heavy traffic demand would emerge. After the anomaly traffic was injected, this work normalized the average traffic to 1 Gbps. This normalization reduces the average traffic volume for the step that does not inject an anomaly since this work sets the average traffic volume that includes anomaly traffic to 1 Gbps. This work applies this option to ARMA, SARIMA, Poisson, and Random models in this evaluation.

Mixed model

This work defines a mixed model consisting of five traffic models and four traffic models with the anomaly option mentioned above. At the beginning of each episode, each user randomly selects one of the nine traffic patterns.

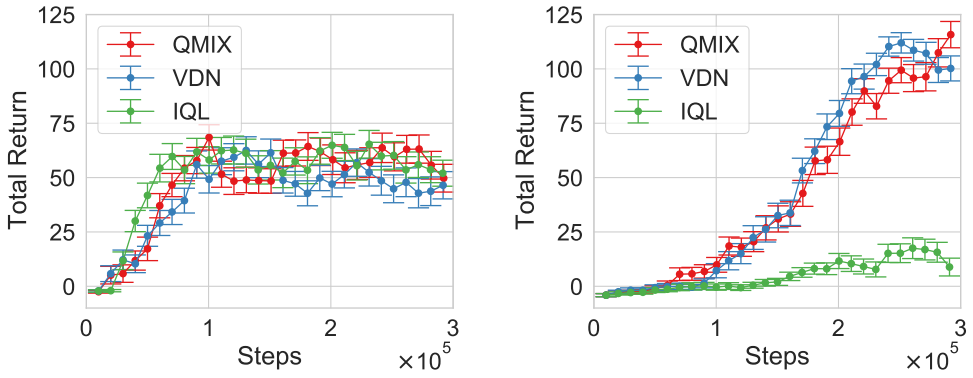
5.3.3 Comparative methods

Table 5.6 summarizes the proposed methods and the comparison methods. QMIX and VDN indicate the Coop-MADRL-based dynamic VN allocation

method shown in Alg. 8, which each use QMIX and VDN in a mixed network and use Eq. (2.7) for end-to-end training. IQL indicates the MADRL-based method without cooperation, i.e., each agent learns on the basis of their reward and Eq. (2.5) is used for end-to-end training. IQL does not impose any restrictions on an agent's action at each time. Note that the previous method of this work [27] restricted one agent that could act at each time, and it is different from IQL. Also note that single-agent DRL was not evaluated because learning is clearly unsuccessful due to requirements for huge training iterations until the Q-values for all actions are sufficiently close to the optimal. When a single agent trains in a practical network, it is estimated that at least 10^{12} training steps will be required even if the agent learned each action once.

Static Allocation (SA) is the heuristic method where VM allocation is fixed to minimize the sum of U_0^L and U_0^S for an average amount of past demands, and routes were dynamically changed as with other methods. Since finding the initial VM allocation for SA is also NP-hard, in this evaluation, this work sequentially decided on the best VM allocation that minimizes the sum of U_0^L and U_0^S for average VN demands consisting of 1 Gbps traffic demand and VM size of 3. By comparing SA and other methods, this work expects to determine the effectiveness of dynamic allocation.

Exhaustive Search (ES) is the meta-heuristic method where the action that maximizes the reward r_t is selected from all candidate actions $\mathcal{A}$ at t. ES finds the best action by exhaustively calculating each reward one by one for all candidate actions at t, which is equivalent to solving an optimization problem every t. The objective of ES is to maximize the immediate reward, which is the reward obtained in the present. Alternatively, the objective of RL is to maximize the delayed reward, which is the expectation of the total rewards to be obtained in the future. By comparing ES and other MADRL-based methods, this work expects to evaluate the difference in performance between the two types of rewards. Note that the solution of ES is different from the global optimal solution because the global optimal allocation is defined as the solution that maximizes the sum of the rewards at each time, as shown in Eq. (5.1). To find the global optimal solution, it is necessary to perfectly predict all future demands and calculate the optimal allocation that maximizes the sum of the rewards at each time. Therefore, the global optimal solution



(a) Simple network.

(b) Practical network.

Figure 5.8: Training curves tracking the agent's total return (training by Mixed model). (©2022 IEEE.)

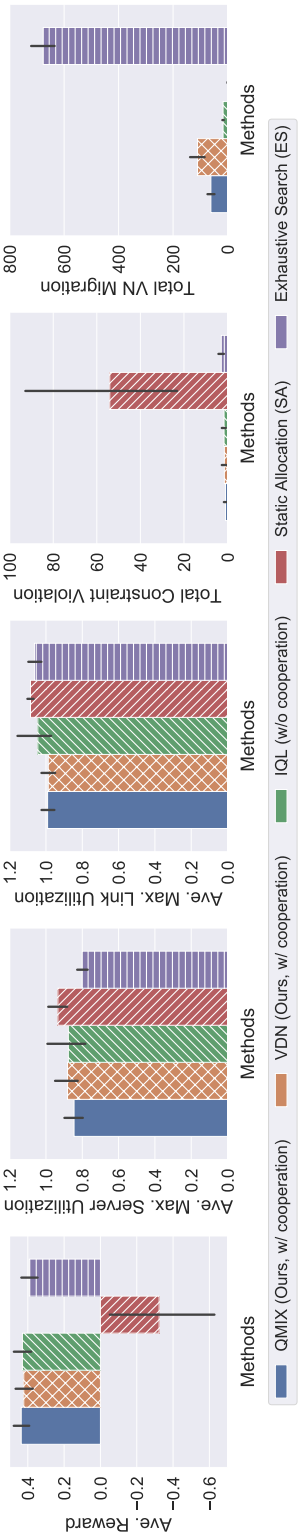
is challenging to calculate within a realistic computation time, even for the simple-network condition.

5.3.4 Evaluation results

Training curve

Figure 5.8 shows the training curves tracking the agent's total return under simple- and practical-network conditions. The total return is defined as the sum of rewards at each time until the end of the episode. This work carried out 5 evaluations for every 1×10^4 steps with random initial conditions. The width of each bar indicates the standard deviation ($\pm\sigma$). This work adopted a mixed model for traffic patterns in training and evaluation.

Figure 5.8(a) shows that the average total return of the three MADRL-based methods increased as the training progressed. It also shows that the curves of the total return of the three methods are almost the same. Figure 5.8(b) shows that the average total return of the Coop-MADRL (QMIX and VDN) based methods increased as the training progresses, while that of the non-Coop-MADRL (IQL) does not increase. This means that IQL cannot learn the suitable allocation that maximizes the objective function while satisfying constraints. This work discusses the performance details in Section 5.3.4.



(a) Training and evaluation by Mixed model, for simple network.

(b) Training and evaluation by Mixed model, for practical network.

Figure 5.9: Performance evaluation for each method. (©2022 IEEE.)

Performance

Figure 5.9 shows the average performance of each method under simple- and practical-network conditions. This work carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations. The width of each bar indicates the standard deviation ($\pm\sigma$). This work used agents trained with the mixed model as described in Fig. 5.8. This work also used the mixed model for the traffic model in this evaluation. The mixed model includes large demand fluctuations by the Random and Poisson models. The performance of each method can roughly be compared with the average reward. This work also investigates the details of performance as shown in the 2–5$^{\text{th}}$ metrics in Figs. 5.9(a) and 5.9(b). That is, the performance metrics are the average maximum server utilization U_t^S, average maximum link utilization U_t^L, total constraint violation, and total VM migration. The server and link utilization usually take a value in the range of 0–1 if all constraints are satisfied. On the other hand, if constraint violations occur at t, the server and link utilization at t may take a value greater than 1 depending on the amount of exceeded resources. When k VMs migrate at a certain time, the total VM migration adds k.

In summary, Coop-MADRL reduced the maximum server and link utilization and drastically reduced the total constraint violations compared with SA. Therefore, the proposed method is effective for simple- and practical-network conditions.

Performance on simple network

This work first compares SA and the other four dynamic allocation methods. Figure 5.9(a) shows that the four dynamic methods performed better than SA. The reason is that the dynamic allocation methods can prevent constraint violations by changing the VN allocation in accordance with the demand change. This also indicates that three MADRL methods performed better than SA and that reducing the action space by using the multi-agent technique is effective. In particular, even IQL performed better than SA. MADRL can reduce the constraint violations without information of the next demand because the RL agent indirectly predicts the next demand and learns the effective VN alloca-

tion change through the relationship between the network state and network efficiency. However, since MADRL calculates rewards on the basis of the current demand and not the next-step demand, constraint violations cannot be absolutely eliminated.

This work next compares ES and the other three MADRL-based methods. ES performed slightly lower than three MADRL-based methods. In addition, the number of total VM migrations is notable in ES, even though ES also considers a VM migration penalty described in Eq. (5.2) as well as other MADRL-based methods. The reason is that, since ES calculates the solution that maximizes the reward for the current demand, not for the next demand as described in Section 5.3.3, the optimal action for ES is frequently changed when the VN demand fluctuates. As a result, VM migration is increased, and its performance worsens. On the other hand, the MADRL-based methods learn a policy that maximizes the expected value of the cumulative reward obtained in the future. Thus, the MADRL-based methods can reduce VM migrations and improve the average reward. The result shows that the optimization for immediate rewards does not necessarily maximize the average reward, and the optimization of delayed reward by RL is effective when demand is fluctuating.

Finally, this work compares three MADRL-based methods. Figure 5.9(a) shows that the performance of the three MADRL-based methods was almost the same. It assumes that the performance of the three methods is saturated because this condition is too simple. This can be seen from the training curves in Fig. 5.8. Although QMIX and VDN perform roughly the same, they differ in terms of total VM migration. This work considers that QMIX can simultaneously reduce the constraint violations and unnecessary VM migrations since agents of QMIX acquired a higher level of cooperation knowledge than those of VDN.

Performance on practical network

Figure 5.9(b) shows the average performance of four of the methods under practical-network conditions. Note that SA changed routes calculated with the route-optimization algorithm at each t, and ES could not evaluate within a practical time because its computation time dramatically increased. Also,

Table 5.7: Average computation time per step for $K = 20$.

Methods	**Simple Network**	**Practical Network**
QMIX (Ours)	2.3 ms	75 ms
VDN (Ours)	2.3 ms	75 ms
IQL	2.3 ms	75 ms
Static Allocation (SA)	1.8 ms	74 ms
Exhaustive Search (ES)	69 s	(estimate) 7.2×10^7 s

note that the average reward was higher for the practical network than for the simple network, which is due to the difference in evaluation conditions, and both reward values are not directly comparable.

Similar to the simple-network condition, three dynamic methods performed better than SA. Comparing its performance under simple- and practical-network conditions, both Coop-MADRL-based methods (QMIX and VDN) performed better than non-Coop-MADRL (IQL) based methods because the degree of freedom of allocation and cooperation improved as the numbers of nodes and links increased. In particular, both cooperative methods were able to reduce the number of total constraint violations drastically. Since each agent in IQL only considers its action without other agents' actions, the agents concentrate on lightly loaded resources in some cases, and this causes violations. On the other hand, QMIX and VDN have higher maximum server utilization than SA. The reason is that agents chose the action that maximized the average reward and minimized the constraint violations even if the maximum server utilization was increased.

Computation time

Table 5.7 shows the average computation time per t of each method, which includes the time to decide the action, update the next state, and evaluate the performance. This work used Intel core i9-9980HK for the evaluation. In the evaluation phase, the computational complexity is the same for the three MADRL methods because the three methods only calculate the forward network of their DNNs without a mixed network calculation in a decentralized manner as described in Section 5.2.1. Under the simple-network conditions,

Table 5.8: Scalability evaluation results for the number of VNs in practical network (QMIX, training by ARMA model).

Number of VNs	**Ave. Reward**
20	0.57 ± 0.10
40	0.47 ± 0.07
50	0.34 ± 0.08
55	0.20 ± 0.24
60	-5.0 ± 0.02

the computation time per t of four methods except ES was less than a few milliseconds, and that of ES drastically increased. MADRL took less than 1 ms for an agent to decide the next action. Though ES was the best method from the viewpoint of performance, it was not effective due to the huge computation time. Under the practical-network conditions, the computation time per t of all methods other than ES was less than one second. The computation times of the four methods except ES were almost equal because the time of the route-optimization calculation became dominant. The computation time of ES is estimated from the computational quantities of iteration since it was difficult to find in conventional time. The optimal solution may be found faster by improving the search algorithm, but it would be difficult to achieve the same speed as MADRL. Finally, this work revealed that the proposed method is also useful as a dynamic VM allocation method in terms of computation time.

This work also mentions the computation time of training agents. For the simple network, QMIX took about 100 minutes and VDN and IQL took about 70 minutes to finish the training of the total time steps $T = 3.0 \times 10^5$. For the practical network, QMIX and VDN took about 7.1 hours and IQL took about 6.5 hours to finish the training of the total time steps $T = 3.0 \times 10^5$. This indicates that the proposed method can learn the optimal VN allocation in less than half a day for networks of the scale used in the evaluation.

Scalability for number of VNs

Table 5.8 shows the scalability of QMIX for the number of VNs K under practical-network conditions. This work carried out 20 calculations with ran-

dom initial conditions and calculated the mean value and the standard deviation of rewards. This work used the ARMA model for the traffic model in training and evaluation. This work increased the number of VNs K until the performance decreased. This work also increased link capacity and server capacity in proportion to K, keeping the average traffic demands and average VM size constant.

As a result, the average reward rapidly decreased when the number of VNs exceeded 60 VNs, and the proposed method was effective up to about 50 VNs. RL should heuristically discover actions that improve the reward and satisfy all constraints in the early learning steps when the agent acts randomly. It seems that scalability could be improved by supporting the initial learning process by providing correct training data by a person. The proposed method is as scalable as the previous methods [25–27] because they simultaneously handled less than 50 VNs at each time step. In this work, a VN is assumed to be a resource-isolated network slice provided by each service provider and shared by many users or IoT devices. This work believes that the proposed method is sufficiently scalable considering the number of services handled by today's networks.

Table 5.9: Computation time for various physical network topologies.

Topology	**# of Nodes**	**# of Links**	**Execution Time**	**LP Time**	**Training time**
Internet2	9	13	0.06 s	0.05 s	0.30 day
Abilene	12	15	0.17 s	0.15 s	0.83 day
Atlanta	15	22	0.43 s	0.39 s	1.6 day
Geant	22	36	3 s	2.7 s	10 day*
France	25	45	5.3 s	4.9 s	18 day*
India35	35	80	21 s	20 s	73 day*
Germany50	50	88	96 s	89 s	3.3×10^2 day*
Ta2	65	108	305 s	286 s	1.0×10^3 day*

*Estimated value

Table 5.10: Computation time for various physical network topologies (with shortest path).

Topology	**# of Nodes**	**# of Links**	**Execution Time**	**Training time**
Geant	22	36	1.6 ms	8.1 h
France	25	45	1.8 ms	16.9 h
India35	35	80	2.0 ms	11.9 h
Germany50	50	88	3.0 ms	10.3 h
Ta2	65	108	3.9 ms	19.8 h

Table 5.11: Performance evaluation results in Atlanta network (training by ARMA model).

Methods	Ave. Reward	Ave. Max. Server Util.	Ave. Max. Link Util.	Constraint Violation	Total VM Migration
QMIX (Ours)	0.43 ± 0.14	0.97 ± 0.39	1.1 ± 0.21	9.5 ± 12	189 ± 62
Static Allocation (SA)	0.37 ± 0.11	0.50 ± 0.07	1.1 ± 0.07	14 ± 13	0.00 ± 0.0

Table 5.12: Performance evaluation results in India35 network (training by ARMA model, with shortest path).

Methods	Ave. Reward	Ave. Max. Server Util.	Ave. Max. Link Util.	Constraint Violation	Total VM Migration
QMIX (Ours)	0.56 ± 0.08	0.90 ± 0.16	0.94 ± 0.28	0.70 ± 1.9	263 ± 29
Static Allocation (SA)	-0.40 ± 0.33	0.45 ± 0.07	1.3 ± 0.08	102 ± 38	0.00 ± 0.0

Table 5.13: Performance evaluation results in Germany50 network (training by ARMA model, with shortest path).

Methods	Ave. Reward	Ave. Max. Server Util.	Ave. Max. Link Util.	Constraint Violation	Total VM Migration
QMIX (Ours)	0.68 ± 0.06	0.84 ± 0.26	0.87 ± 0.27	0.10 ± 0.45	118 ± 32
Static Allocation (SA)	-0.78 ± 0.22	0.45 ± 0.07	1.5 ± 0.12	142 ± 27	0.00 ± 0.0

Table 5.14: Performance evaluation results in Ta2 network (training by ARMA model, with shortest path).

Methods	Ave. Reward	Ave. Max. Server Util.	Ave. Max. Link Util.	Constraint Violation	Total VM Migration
QMIX (Ours)	0.53 ± 0.26	0.87 ± 0.32	1.1 ± 0.67	2.8 ± 10	174 ± 44
Static Allocation (SA)	-1.3 ± 0.19	0.45 ± 0.07	1.9 ± 0.19	181 ± 16	0.00 ± 0.0

Network topology dependency

To verify the effectiveness of the proposed method for larger physical network topologies, this work evaluates the computation time and performance of the proposed method for various network topologies. The network topology used in the evaluation refers to SNDLib [84], which is a library of test instances of survivable fixed telecommunication network design. In all evaluations, this work used QMIX as the learning algorithm and the ARMA model as the traffic model, and this work sets the number of VNs to $K = 20$. This work also randomly sets user nodes and server nodes for all topologies. Other conditions are the same as in the above evaluation.

Table 5.9 shows the computation time for various physical network topologies. The execution time shows the computation time required to select the agent's action, update the VN allocation, and calculate the reward in each execution step, which corresponds to lines 3–7 in Alg. 8. The LP time shows the computation time required to calculate the route-optimization algorithm in each execution step, which is a part of execution time and corresponds to line 4 in Alg. 9. The training time shows the computation time required for the agents to complete the training of the total time steps $T = 3 \times 10^5$. The result shows that all computation times drastically increase as the number of topology nodes increases. The result also shows that most of the execution and training time is spent solving the route-optimization problem. Due to the drastic increase in computation time, this work estimated the training time for large topologies based on the execution time and the total training steps. Here, this work assumes that the total training steps $T = 3 \times 10^5$ are sufficient for all topologies, but larger ones may require more training steps.

This work considers the practical limits of topology size in terms of computation time. The execution time determines the VN allocation control interval. For example, when the execution time takes 5 minutes, the control interval cannot be shorter than 5 minutes. If we aim to keep the control interval under one minute, the limit of the number of nodes can be estimated to be about 30 nodes. Similarly, if we aim for a control interval of 5 minutes or less, the limit can be estimated to be about 50 nodes. As for the training time, assuming that the reasonable training time is less than one week, the limit of the number of

nodes can be estimated to be about 20 nodes. In conclusion, unless speeding up the computation time, this work can estimate that the upper limit of the number of nodes to which the proposed method can be applied is about 20. Parallelization is one of the promising candidates for speedup the proposed method. Existing studies have reported that a distributed DRL can dramatically speed up the agent training process. For example, Espeholt *et al.* [85] developed a distributed DRL architecture that scales up to thousands of machines without sacrificing data efficiency or resource utilization. The proposed method may be applied to up to 30 nodes by parallelizing the agent training process.

To verify the effectiveness of the proposed method for network topologies within 20 nodes, this work evaluated the performance of the proposed method on Atlanta network with 15 nodes and 22 links. Figure 5.10 shows the training curves tracking the agent's total return for Atlanta network. As in the practical-network condition, this work sets each server capacity to 30 and each link capacity to 3.6 Gbps. The result shows that the average total return of QMIX increased as the training progressed. Table 5.11 shows the average performance of QMIX and SA. Note that, as described in Section 5.3.3, SA also dynamically selects an optimal routes calculated with the route-optimization algorithm at each t. The result shows that QMIX performed better than SA because QMIX can prevent constraint violations by changing the VN allocation by the demand change. Although this work can confirm the effectiveness of QMIX, this work found that the relative performance improvement decreases compared to the result of Fig. 5.9(b). This work considers two reasons for the decrease in the performance difference between the two methods. First, this work supposes that the larger network topology makes dynamic routing more effective, and the dynamic routing improved the performance of VN allocation regardless of VM placement. In other words, the dynamic routing could compensate for the performance degradation of SA caused by the inefficient VM placement. Next, this work also supposes that the larger network topology has increased the required training steps. If so, this work can improve the performance of QMIX by increasing the training steps.

Next, based on the above results, this work discusses applying the proposed method for larger network topologies by drastically reducing computation time.

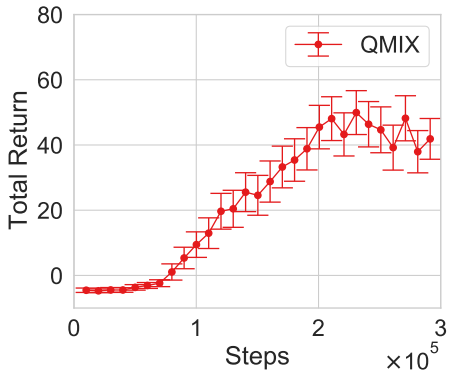
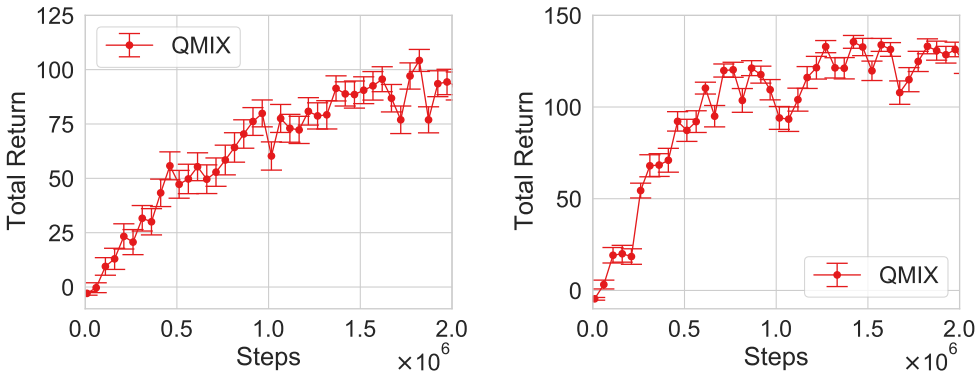


Figure 5.10: Training curves tracking the agent's total return in Atlanta network. (©2022 IEEE.)



(a) India35. (b) Germany50.

Figure 5.11: Training curves tracking the agent's total return with the proposed method using the shortest path. (©2022 IEEE.)

Since most of the computation time is spent solving the route-optimization problem as shown in Table 5.9, relaxing the route-optimization problem and replacing dynamic path allocation with static path allocation is an effective way of speeding up the method. Specifically, this work replaces the optimal path by LP with the shortest path by Dijkstra's algorithm, which corresponds to line 4 in Alg. 9. This work then evaluates the computation time and performance for various network topologies. Finally, this work shows that the proposed method using the shortest path is adequate for practical topology size.

Table 5.10 shows the computation time in the proposed method using the

shortest path for various physical network topologies. To ensure sufficient training, this work increased the total time steps to $T = 2 \times 10^6$. The results show that all computation times are drastically faster than those estimated in Table 5.9. For all conditions, the execution time is less than one second, and the training time is less than one day. Moreover, since Ta2 network is the largest topology that mimics the core network in SNDLib, this work concludes that the proposed method using the shortest path has no limitation on the number of nodes in the range of practical network topologies. The result also shows that the training time does not depend on the number of nodes. The reason is that other parameters, e.g., physical network and VN request parameters, are more dominant factors in determining the training time than the number of nodes.

Figure 5.11 shows the training curves tracking the agent's total return for India35 network and German50 network. This work sets each server capacity to 30 and each link capacity to 5.0 Gbps. This work increased link capacities because the shortest path concentrates the traffic load on some links. The result shows that the average total return of QMIX increased as the training progressed. Tables 5.12–5.14 show the average performance of QMIX and SA. Both QMIX and SA used the shortest path for route calculation, since solving LP at each step is difficult for both methods due to the computation time. The results show that QMIX performed significantly better than SA. This performance difference is since QMIX dynamically changes the placement of VMs or not. In particular, the performance of SA dramatically decreased due to static routing. The reason is that, as mentioned before, the performance of SA was strongly dependent on dynamic routing. In conclusion, this work revealed that the proposed method using the shortest path performs sufficiently in large network topologies.

In summary, this work revealed that the proposed method could be applied to network topologies with less than 65 nodes. More precisely, the proposed method with route-optimization could be applied to network topologies with less than 20 nodes. The proposed method that uses the shortest path for route calculation could be applied regardless of the topology size. In addition, this work revealed that the proposed method outperforms SA in terms of performance regardless of the topology size.

Table 5.15: Evaluation of average rewards for various traffic models in simple network (training by Mixed model).

Traffic Models	QMIX, Mix	VDN, Mix	IQL, Mix	Static Allocation (SA)
ARMA	0.46 ± 0.11	0.46 ± 0.12	0.15 ± 0.77	-0.22 ± 0.68
ARMA with other parameters	0.44 ± 0.12	0.43 ± 0.15	0.36 ± 0.41	-0.32 ± 0.71
SARIMA	0.43 ± 0.11	0.42 ± 0.11	0.35 ± 0.25	-0.30 ± 0.73
Poisson	0.36 ± 0.11	0.35 ± 0.12	0.35 ± 0.14	-0.41 ± 0.73
Random	0.23 ± 0.82	0.23 ± 0.74	0.31 ± 0.55	-0.26 ± 0.73
ARMA with anomaly	0.39 ± 0.13	0.38 ± 0.13	0.32 ± 0.43	-0.36 ± 0.72
SARIMA with anomaly	0.45 ± 0.09	0.44 ± 0.09	0.43 ± 0.11	-0.38 ± 0.75
Poisson with anomaly	0.46 ± 0.11	0.44 ± 0.15	0.34 ± 0.53	-0.10 ± 0.60
Random with anomaly	0.39 ± 0.13	0.38 ± 0.13	0.32 ± 0.43	-0.36 ± 0.72
Mix All Traffic	0.44 ± 0.10	0.42 ± 0.11	0.43 ± 0.11	-0.32 ± 0.70
Ave. All Traffic Models	0.41 ± 0.28	0.40 ± 0.26	0.34 ± 0.42	-0.30 ± 0.70

Table 5.16: Evaluation of average rewards for various traffic models in practical network (training by Mixed model).

Traffic Models	QMIX, Mix	VDN, Mix	IQL, Mix	Static Allocation (SA)
ARMA	0.67 ± 0.07	0.58 ± 0.08	0.20 ± 0.32	0.20 ± 0.21
ARMA with other parameters	0.70 ± 0.04	0.61 ± 0.06	0.23 ± 0.24	0.31 ± 0.14
SARIMA	0.67 ± 0.05	0.60 ± 0.07	0.19 ± 0.20	0.16 ± 0.20
Poisson	0.66 ± 0.05	0.58 ± 0.05	0.11 ± 0.32	0.08 ± 0.13
Random	0.67 ± 0.05	0.58 ± 0.07	0.15 ± 0.36	0.13 ± 0.12
ARMA with anomaly	0.66 ± 0.04	0.57 ± 0.05	0.22 ± 0.15	0.07 ± 0.10
SARIMA with anomaly	0.69 ± 0.05	0.61 ± 0.06	0.22 ± 0.16	0.13 ± 0.19
Poisson with anomaly	0.72 ± 0.06	0.63 ± 0.06	0.30 ± 0.18	0.27 ± 0.14
Random with anomaly	0.66 ± 0.04	0.57 ± 0.05	0.22 ± 0.15	0.07 ± 0.10
Mix All Traffic	0.68 ± 0.06	0.60 ± 0.06	0.26 ± 0.15	0.15 ± 0.12
Ave. All Traffic Models	0.68 ± 0.06	0.59 ± 0.06	0.21 ± 0.24	0.16 ± 0.17

Table 5.17: Evaluation of average rewards for various traffic models in practical network (training by each model).

Traffic Models	QMIX, ARMA	QMIX, ARMA w/ ano.	QMIX, Poisson	QMIX, Random	QMIX, Mix
ARMA	0.57 ± 0.10	0.58 ± 0.13	0.69 ± 0.06	0.63 ± 0.07	0.67 ± 0.07
ARMA with other parameters	0.59 ± 0.08	0.65 ± 0.08	0.72 ± 0.07	0.67 ± 0.06	0.70 ± 0.04
SARIMA	0.56 ± 0.09	0.62 ± 0.11	0.69 ± 0.05	0.65 ± 0.06	0.67 ± 0.05
Poisson	0.43 ± 0.13	0.56 ± 0.14	0.65 ± 0.06	0.60 ± 0.05	0.66 ± 0.05
Random	0.42 ± 0.09	0.59 ± 0.13	0.67 ± 0.07	0.62 ± 0.07	0.67 ± 0.05
ARMA with anomaly	0.44 ± 0.08	0.60 ± 0.10	0.67 ± 0.04	0.62 ± 0.06	0.66 ± 0.04
SARIMA with anomaly	0.55 ± 0.08	0.62 ± 0.08	0.69 ± 0.06	0.64 ± 0.05	0.69 ± 0.05
Poisson with anomaly	0.59 ± 0.05	0.64 ± 0.12	0.73 ± 0.05	0.69 ± 0.05	0.72 ± 0.06
Random with anomaly	0.44 ± 0.08	0.60 ± 0.10	0.67 ± 0.04	0.62 ± 0.06	0.66 ± 0.04
Mix All Traffic	0.52 ± 0.05	0.58 ± 0.12	0.69 ± 0.05	0.65 ± 0.05	0.68 ± 0.06
Ave. All Traffic Models	0.51 ± 0.11	0.60 ± 0.11	0.69 ± 0.06	0.64 ± 0.06	0.68 ± 0.06

Generalization performance

The generalization performance is a measure of how accurately an algorithm is able to perform outcomes for previously unseen data. To evaluate the generalization performance for traffic demands, this work evaluated the average reward using various traffic models that were different from those during training. Tables 5.15 and 5.16 show the average rewards of each method when evaluated with various traffic models under simple- and practical-network conditions. This work carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations. This work used agents trained with the mixed model as described in Fig. 5.8. This work also used 10 types of various traffic models for evaluation as described in the first columns of Tables 5.15 and 5.16. The row "Mix all Traffic" corresponds to the results in Fig. 5.9. The row "Ave. all traffic models" means the average results of 10 types of traffic models.

For simple-network conditions, QMIX and VDN performed better than IQL for the average of all traffic models. SA also performed lower than other dynamic allocation methods. In QMIX and VDN, since the mixed model's performance and the average of all model's performance are approximately equal, training with the mixed model was effective for many traffic patterns. Moreover, the performance of QMIX and VDN does not decrease even for traffic models with the anomaly. Therefore, QMIX and VDN training with the mixed model obtains a generalization performance for traffic demands. In contrast, QMIX and VDN underperformed in the Poisson and Random models because these models include the traffic patterns which were not experienced. IQL sufficiently performed on the mixed traffic model but unsuccessfully performed on other traffic models. This shows that QMIX and VDN learn more superior policies through cooperative learning.

For practical-network conditions, the results are mostly the same as those under simple-network conditions. As mentioned in Section 5.3.4, though the average reward for the practical network was higher than that for the simple network, both reward values are not directly comparable. Comparing their performances under simple- and practical-network conditions, QMIX and VDN for the Poisson and Random models improved. This work considers that, since

freedom of allocation improved as the numbers of nodes and links increased in the practical network, more flexible allocation has become possible even when traffic fluctuates.

Next, this work comprehensively analyzed the relationship between the generalization performance for traffic demands and the traffic models used during training. This evaluation can determine the best traffic models used during training for obtaining a generalization performance for traffic demands. Table 5.17 shows the average rewards of QMIX when this work used various traffic models for training and evaluation under practical-network conditions. This work adopted five types of traffic models for training: ARMA, ARMA with the anomaly, Poisson, Random, and Mixed model. As in the evaluation in Table 5.16, this work carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations.

When the ARMA model was used for training, the performance of QMIX did not decrease with the ARMA model with other parameters and the SARIMA model. The performance for some models with the anomaly option was not decreased because of the normalization in the traffic generation process as described in Section 5.3.2. However, the performance decreased with unknown traffic, such as the Poisson model, the Random model, and models with anomaly options.

When the ARMA model with an anomaly was used for training, all traffic models performed better than when the ARMA model was used for training. This is because agents experience more traffic changes and learn how to act to avoid congestion when anomaly traffic occurs in advance.

When the Poisson, Random, and mixed models were used for training, QMIX performed outstandingly in all evaluation conditions. Moreover, when the ARMA model was used for evaluation in the first row of Table 5.17, the average reward for training with the mixed model was higher than that for training with the ARMA model. Therefore, this work concludes that training with the mixed model is suitable for obtaining high generalization performance for traffic demands. This work considers that the reason is that the mixed model contains various traffic fluctuation patterns. While the mixed model performs well, each traffic model needs to be prepared so that it is composed of a part of a mixed model and assumes user traffic patterns when a mixed

model for training is generated. On the other hand, interestingly, the Poisson model for training performed equal to or better than the mixed model. This work assumes the reason to be that this model also contains various traffic changes. Since the Poisson model is easy to generate, training with this model will always be beneficial if it leads to a high generalization performance for various traffic demands. To analyze the results in more detail, it is necessary to solve the interpretability problem of DRL, which is an unsolved challenge in machine learning theory and is a future work of this work.

5.3.5 Discussion

This work discusses the future work of the proposed method. This work revealed the effectiveness of the proposed method, but there are still important challenges that should be researched for applying the proposed method in a commercial environment.

A major challenge is the generalization of agent training, i.e., an agent trained in one environment might not perform in other environments not experienced during training. For example, when changing user placement, the number of VNs, and the physical network topology, or when observing new traffic patterns, the agent should retrain from the beginning. This is a general challenge in machine learning, not unique to DRL-based dynamic allocation. As mentioned in Section 5.3.4, since the proposed method takes a long time to train agents, such as a few hours or a day, the re-training could be a significant problem when the network conditions are frequently changed. While this work shows that training by the mixed or Poisson model achieves generalization performance for traffic demands, whole new traffic patterns will possibly emerge with the development of 5G and other technologies in the future. One potential solution is grouping VNs, which can keep the number of VN groups constant (e.g., using [27]). This grouping will also lead to improving scalability. Another solution is aggregating all traffic to a small number of flows and handling only predictable flow patterns (e.g., using [86]). Although these grouping and aggregation are effective to limit the demand patterns, the performance of allocation will be decreased due to coarse-grained allocation. The other solution is transfer learning, which focuses on storing knowledge gained

while solving one domain and applying it to a different domain so that it can be learned efficiently.

This work also discusses the challenge of interpretability of machine learning related to the challenge of generalization of machine learning. Since the DNN model included in the DRL is a black box, it is difficult to explain why the DNN outputs the results. For example, in this work, it is impossible to understand why the Poisson model achieved the best performance. When applying these DRL-based techniques in actual network operations, the challenge also arises that the operator cannot trust the agents' outputs.

The other challenge is the ideal migration. As mentioned in Section 5.1.1, the proposed method assumes ideal VM migration, i.e., the system can immediately reallocate VM without interrupting the running service. Note that the proposed method can be applied even if the migration takes a long time. However, when migration becomes a bottleneck, the dynamic allocation has difficulty following the demand fluctuations. One possible solution that a control algorithm can contribute is developing a more realistic penalty function of VM migration taking into account the availability and sustainability of the service (e.g., using [87]).

5.4 Chapter summary

This chapter proposed a dynamic virtual network (VN) allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly optimize the network resources even when traffic demands change drastically by applying MADRL for dynamic VN allocation. It can also reduce the agents' constraint violations such as network congestion and server overload and reduce the reallocation such as virtual machine (VM) migration by introducing a cooperative element for MADRL. Simulations revealed that the proposed dynamic VN allocation method can reduce the maximum server and link utilization and drastically reduce the constraint violations compared with that of a static VN allocation method under practical-network conditions. In contrast, the evaluation also revealed that the Exhaustive Search (ES) that maximizes the reward at each time does not necessarily maximize the average rewards when the traffic demands fluctuate. Moreover, the com-

putation time of the proposed method was less than one second, which is significantly shorter than that of ES. As a result, this work revealed that the proposed method simultaneously enables efficient and immediate dynamic VN allocation. Finally, this work evaluated the generalization performance for various traffic demands. The results revealed that the agent training with mixed various traffic models could achieve a high generalization performance for all traffic models.

For future work, this work plans to evaluate the performance of the proposed method in more complicated use-cases, e.g., service function chaining (SFC), in real-world demands and applications, and a test-bed environment. This work also plans to improve the interpretability of deep reinforcement learning (DRL) by analyzing the relationship between the network state, the agent's actions, and the allocation results in detail. Moreover, this work plans to evaluate the methods involving the approach described in Section 5.3.5.

Chapter 6

Cooperative multi-agent deep reinforcement learning for task offloading

Edge computing is a new paradigm to provide computing capability at the edge servers close to end devices. A significant research challenge in edge computing is finding efficient task offloading to edge and cloud servers considering various task characteristics and limited network and server resources. Several reinforcement learning (RL)-based task-offloading methods have been developed, because RL can immediately output efficient offloading by pre-learning. However, these methods do not take into account clouds or focus only on a single cloud. They also do not take into account the bandwidth and topology of the backbone network. Such shortcomings strongly limit the range of applicable networks and degrade task-offloading performance. Therefore, this work formulates a task-offloading problem for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. This work also proposes a task-offloading method that is based on cooperative multi-agent deep RL (Coop-MADRL). A part of this work in this chapter was presented in [88, 89]. This method introduces a cooperative multi-agent technique through centralized training and decentralized execution, improving task-offloading efficiency. Simulations revealed that the proposed method can minimize network utilization and task latency while minimizing constraint violations in less than one

millisecond in various network topologies. It also shows that cooperative learning improves the efficiency of task offloading. This work demonstrated that the proposed method has generalization performance for various task types by pre-training with many resource-consuming tasks.

This chapter is structured as follows. Section 6.1 describes the formulation of the task-offloading problem. Section 6.2 describes the proposed Coop-MADRL-based task-offloading method. Section 6.3 describes the evaluation of its performance, and Section 6.4 concludes the chapter.

6.1 Problem formulation

6.1.1 Overview

This work describes an overview of the task-offloading problem. This work assumes a network consisting of EDs, edges, and clouds. EDs generate various tasks with diverse applications. Each task consists of the required computing demand, traffic demand, and maximum permissible latency to accomplish the task. Each ED can compute its tasks locally or offload tasks to the neighboring edge or cloud. This work collectively refers to edges and clouds as nodes. All nodes have computing resources to execute the tasks instead of EDs, called edge or cloud servers. All nodes also have a function of traffic forwarding to another node as a router. Only each edge has a function that determines the optimal node to offload each task. Each cloud cannot offload tasks to other nodes, only execute the tasks. This work assumes that the ED determines whether to offload its tasks; optimizing the ED's decision is outside the scope of this work. This work aims to optimize the offloaded server and the route between ED and server for the accepted task.

This work describes the procedure of the proposed task-offloading method. This work considers a discrete time-step t and assumes that each ED has one or more tasks and consider $\mathcal{K}$ tasks during $t \in [0, T]$. At the beginning of each t, each task arrives at the nearest edge of each ED. Each edge observes the information about tasks accepted at each edge and the overall network usage of all nodes and links. On the basis of the observation, the proposed method deployed in each edge calculates the optimal node to offload the task

Table 6.1: Notation definitions for physical-network model.

Notation	**Definition**
$G(\boldsymbol{N}, \boldsymbol{L})$	Network graph
$n \in \boldsymbol{N}$	Node
link $(i, j) \in \boldsymbol{L}$	Link from node i to node j
$e \in \boldsymbol{E} \subset \boldsymbol{N}$	Edge
$c \in \boldsymbol{C} \subset \boldsymbol{N}$	Cloud
v_i^N	i^{th} node-processing capability
w_i^N	i^{th} node capacity
w_{ij}^L	(i, j) link capacity
α_{ij}^L	Propagation latency of link (i, j)

(see Section 6.2 for details). The offloading node is calculated only once, and the neighboring edges that receive the tasks cannot offload it again to the other edge or cloud. When multiple tasks arrive simultaneously on each edge at t, the proposed method repeats determining the offloading node in a first-in-first-out (FIFO) manner. The method then aggregates the traffic-demand information between nodes and calculates and updates the optimal route between nodes. Next, each edge forwards tasks to the optimal nodes through the optimal route, and the node executes the task and returns the result to the ED. The upload and download routes of each task may differ since the proposed method handles the upload and download traffic as individual traffic demands. After a certain amount of time, it proceeds to the next $t + 1$. The executing tasks continue to consume the resources of the offloaded node and the link(s) it traverses until it returns a result to the ED. The tasks accepted at t do not need to be completed before $t + 1$. Note that it is also adequate to calculate and update the route once every several steps. In this case, the proposed method calculates the optimal route on the basis of the average traffic volumes between nodes for several steps.

6.1.2 Network model

Table 6.1 summarizes the notation definitions of the physical-network model. This work considers the physical-network graph $G(\boldsymbol{N}, \boldsymbol{L})$ consisting of a phys-

Table 6.2: Notation definitions for task model.

Notation	Definition
$t \in \boldsymbol{T}$	Time-step ($\boldsymbol{T}$: Total time steps)
$\mathcal{D} := \{\boldsymbol{D}_k\}$	Task set ($k \in \mathcal{K}$, $\mathcal{K}$: Total tasks)
t_k	Acceptance time of k^{th} task
β_k	Task type of k^{th} task
C_k	Computing demand of k^{th} task
$B_k^{\text{up}}, B_k^{\text{down}}$	Upload/Download traffic demand of k^{th} task
$\tau_k^{\max}$	Maximum permissible latency of k^{th} task

ical node set $\boldsymbol{N}$ and physical link set $\boldsymbol{L}$. This work assumes that each physical node has a role as an edge or cloud. This work also assumes that EDs connect to the nearest edge through the access network, which is not included in $\boldsymbol{G}(\boldsymbol{N}, \boldsymbol{L})$. This work also assumes that the routes in the access network take the shortest path, and route optimization in the access network is outside the scope of this work.

This work denotes the edge as $e \in \boldsymbol{E} \subset \boldsymbol{N}$ and the cloud as $c \in \boldsymbol{C} \subset \boldsymbol{N}$. This work also denotes the numbers of nodes, edges, and clouds as $|\boldsymbol{N}|$, $|\boldsymbol{E}|$, and $|\boldsymbol{C}|$, respectively. This work denotes the node-processing capability of the i^{th} node as $v_i^N \in \mathbb{R}^+$, which indicates the limit of computing resources, e.g., the CPU capability per second in the i^{th} node ([G cycles/s]). Here, $\mathbb{R}^+$ indicates the set of positive real numbers. This work also denotes the node capacity of the i^{th} node as $w_i^N \in \mathbb{N}$, which indicates the upper limit of the number of allocated tasks. This work assigns one CPU core to each task, i.e., w_i^N equals the number of CPU cores in the i^{th} node. This work denotes the bandwidth capacity of link (i, j) as $w_{ij}^L \in \mathbb{R}^+$, which indicates the limit of bandwidth resources ([Mbps]). All links also have propagation latency depending on the distance between each node. This work defines the propagation latency of link (i, j) as $\alpha_{ij}^L \in \mathbb{R}^+$ ([ms]). A concrete example is shown in Fig. 6.1.

6.1.3 Task model

Table 6.2 summarizes the notation definitions of the task model. This work describes a task model for uniformly representing various types of tasks of

EDs. This work defines the task set as $\mathcal{D} = \{\boldsymbol{D}_k\}$ and the k^{th} task as $\boldsymbol{D}_k := [t_k, \beta_k, C_k, B_k^{\text{up}}, B_k^{\text{down}}, \tau_k^{\max}]$. Here, $t_k \in T$ is the accepted time of the k^{th} task, $\beta_k \in \mathbb{N}$ is the task type uniquely given for each application, C_k is the required computing demand ([G cycles]), $B_k^{\text{up}} \in \mathbb{R}^+$ and $B_k^{\text{down}} \in \mathbb{R}^+$ are the required upload and download traffic demands ([Mbits]), and $\tau_k^{\max} \in \mathbb{R}^+$ is the maximum permissible latency to accomplish the k^{th} task ([ms]). Regarding the relationship between t_k and $\tau_k^{\max}$, if this work can complete the processing of the task k by the time $t_k + \tau_k^{\max}$, this work satisfies the requirements of the task k. The computing demand refers to the total amount of CPU cycles needed to accomplish the task. The traffic demand refers to the amount of traffic generated by the task. The reason for distinguishing between upload and download traffic is to accommodate tasks, the traffic volume of which changes before and after the computation process on the assigned server. When $\boldsymbol{D}_k$ is accepted, it consumes the computing and bandwidth resources on $\boldsymbol{G}(\boldsymbol{N}, \boldsymbol{L})$ depending on the amount of computing and traffic demand of $\boldsymbol{D}_k$. If the task is assigned to the nearest edge of the ED, the bandwidth resources consumed on $\boldsymbol{G}(\boldsymbol{N}, \boldsymbol{L})$ are regarded as 0. Note that if the task parameters are unknown, other systems need to be used to estimate the parameters, which is out of the scope of this work.

6.1.4 Optimization problem

This work formulates the optimal task-offloading problem, which aims to minimize Eq. (6.1) while satisfying the constraints in Eqs. (6.2)–(6.14). Table 6.3 summarizes the notation definitions of the task-offloading problem. This work aims to find optimal task-allocation variables $\boldsymbol{Y}$ and routing variables $\boldsymbol{X}_t$. Here, $\boldsymbol{Y} := \{y_{kn}\}$ shows the task allocation in which y_{kn} is 1 if the computing demand of the k^{th} task is assigned to the n^{th} node; otherwise, 0. The $\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$ shows the proportion of traffic demand m_t^{pq} from the origin node p to destination node q passing through link (i, j) at t. Here, $\boldsymbol{M}_t := \{m_t^{pq}\}$ shows the traffic demand matrix between nodes p and q at t. The m_t^{pq} represents the total traffic demand between nodes p and q at t. The nodes p and q in m_t^{pq} are determined by device placements and task-allocated servers. A detailed formulation is shown in Eqs. (6.12)–(6.13). Since this work assumes multi-path

Table 6.3: Notation definitions for task-offloading problem.

Notation	**Definition**
$\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$	Proportion of passed m_t^{pq} on link (i, j)
$\boldsymbol{Y} := \{y_{kn}\}$	Task allocation (task k, node n)
$\boldsymbol{M}_t := \left\{m_t^{pq}\right\}$	Traffic matrix from node p to node q
$\boldsymbol{Z} := \{z_{ke}\}$	Device placement (task k, edge e)
$\boldsymbol{U}_t^N := \left\{u_{i,t}^N\right\}$	i^{th} node utilization at step t
$\boldsymbol{U}_t^L := \left\{u_{ij,t}^L\right\}$	(i, j) link utilization at step t
$U_t^N = \max_i \left(\boldsymbol{U}_t^N\right)$	Maximum node utilization at step t
$U_t^L = \max_{ij} \left(\boldsymbol{U}_t^L\right)$	Maximum link utilization at step t
τ_k^N, τ_k^{RTT}	Node latency and RTT latency of k^{th} task
λ	Weighting parameter of objective function
$\mathbb{I}_{k,t}$	Binary variable indicating executing tasks
$\mathcal{P}_k^{\text{up}}$, $\mathcal{P}_k^{\text{down}}$	Upload and download path set of k^{th} task
τ_p	Latency coefficient of path p
$\mathcal{L}_p$	Link set along with path p
r_p	Traffic-splitting ratio of path p

routing, $x_{ij,t}^{pq}$ can take a continuous value between 0 and 1. This work also defines the ED placement as $\boldsymbol{Z} := \{z_{ke}\}$, in which z_{ke} is 1 if the nearest edge of the ED requested k^{th} task is the e^{th} edge; otherwise, 0. This work assumes that $\boldsymbol{Z}$ is constant during $t \in [0, T]$ to ignore the effects of ED movement, e.g., handover. In other words, this work assumes that the ED requesting to process the task at edge e stays near edge e for the processing time, e.g., at least a few seconds.

This work introduces the objective function:

$$\min : \sum_{t \in T} \left(U_t^N + U_t^L\right) + \lambda \sum_{k \in \mathcal{K}} \left(\frac{\tau_k^N + \tau_k^{\text{RTT}}}{\tau_k^{\max}}\right) / |\mathcal{K}|, \tag{6.1}$$

where U_t^N and U_t^L show the maximum node and link utilization at t, which are respectively defined as $U_t^N := \max_i(\boldsymbol{U}_t^N)$ and $U_t^L := \max_{ij}(\boldsymbol{U}_t^L)$. This work denotes the i^{th} node utilization as $u_{i,t}^N \in \boldsymbol{U}_t^N$ and the link (i, j) utilization as $u_{ij,t}^L \in \boldsymbol{U}_t^L$. The terms τ_k^N and τ_k^{RTT} show the node and round-trip time (RTT)

latency of the k^{th} task. RTT is the time it takes to get from the source to the destination and back from the destination to the source. Their definitions and formulations are described in the subsection on the latency constraints. The term λ indicates the weighting parameter determining the importance ratio of resource efficiency and task latency.

This work imposes three types of constraints: node capacity, link capacity, and task latency. This work first defines the binary variable $\mathbb{I}_{k,t}$ as follows:

$$\mathbb{I}_{k,t} := \begin{cases} 1 & (t_k \leq t \leq t_k + \tau_k^N + \tau_k^{\text{RTT}}) \\ 0 & (\text{otherwise}), \end{cases} \tag{6.2}$$

where $\mathbb{I}_{k,t}$ is 1 if the k^{th} task is executing at t; otherwise, 0. Here, t_k is the acceptance time of the k^{th} task and $t_k + \tau_k^N + \tau_k^{\text{RTT}}$ shows the completion time of the k^{th} task. The total latency $\tau_k^N + \tau_k^{\text{RTT}}$ of the k^{th} task can be modeled by Eqs. (6.15)–(6.16) (details below). Latency in a real-world environment does not necessarily occur as modeled. However, in a real-world environment, this work can measure latency and can use measured values instead of the modeled equations.

Node-capacity constraints

A task-allocation variable y_{kn} is formulated to minimize the maximum node utilization U_t^N while satisfying the node-capacity constraints as follows:

$$\text{s.t.} \quad : \quad \sum_{n \in \boldsymbol{N}} y_{kn} = 1 \qquad (\forall k \in \mathcal{K}) \tag{6.3}$$

$$\sum_{k \in \mathcal{K}} \mathbb{I}_{k,t}\, y_{kn} \leq w_n^N U_t^N \qquad (\forall n \in \boldsymbol{N}) \tag{6.4}$$

$$y_{kn} \in \{0, 1\} \qquad (\forall k \in \mathcal{K}, \forall n \in \boldsymbol{N}) \tag{6.5}$$

$$0 \leq U_t^N \leq 1. \tag{6.6}$$

Equation (6.3) shows that the computing demand of each task must be allocated to any node. Equation (6.4) shows the constraint of node capacity. The term $\mathbb{I}_{k,t}\, y_{kn}$ in Eq. (6.4), the product of two binary variables, is 1 if the k^{th} task is allocated to node n and is executing at t, because $\mathbb{I}_{k,t}\, y_{kn} = 1$ if $\mathbb{I}_{k,t} = 1$ and $y_{kn} = 1$. By calculating the sum over k, the left side of Eq. (6.4) shows the number of tasks executing at node n at t. The left side of Eq. (6.4) shows

the n^{th} node capacity at t when the maximum node utilization is U_t^N. In other words, it shows the maximum number of tasks that the n^{th} node can perform at t. This work imposes the upper bound $w_n^N U_t^N$, which is stronger than w_n^N. Equations (6.5)–(6.6) show the range of variables.

Link-capacity constraints

This work formulates the link-capacity constraints as a multi-commodity flow problem, which is a network flow problem with multiple commodities (i.e., traffic flow demands) between source and destination nodes. These equations give the capacity and flow conservation constraints that the routing variables must satisfy. If we find routing variables that pass through the shortest path, we can use the shortest path algorithm, such as Dijkstra or Bellman-Ford. However, these algorithms do not take capacity constraints into account. If all traffic demands pass through the shortest path, traffic may concentrate on a single link, causing congestion.

A routing variable $x_{ij,t}^{pq}$ is formulated to minimize the maximum link utilization U_t^L while satisfying the link-capacity constraints as follows:

$$\text{s.t.} \quad : \quad \sum_{j:(i,j)\in L} x_{ij,t}^{pq} - \sum_{j:(j,i)\in L} x_{ji,t}^{pq} = 0 \tag{6.7}$$

$$(\forall p, q \in \boldsymbol{N}, i \neq p, i \neq q)$$

$$\sum_{j:(i,j)\in L} x_{ij,t}^{pq} - \sum_{j:(j,i)\in L} x_{ji,t}^{pq} = 1 \tag{6.8}$$

$$(\forall p, q \in \boldsymbol{N}, i = p)$$

$$\sum_{p,q\in \boldsymbol{N}} m_t^{pq} x_{ij,t}^{pq} \leq w_{ij}^L U_t^L \tag{6.9}$$

$$(\forall (i,j) \in L, \forall p, q \in \boldsymbol{N})$$

$$0 \leq x_{ij,t}^{pq} \leq 1 \quad (\forall (i,j) \in L, \forall p, q \in \boldsymbol{N}) \tag{6.10}$$

$$0 \leq U_t^L \leq 1. \tag{6.11}$$

Equations (6.7)–(6.8) show the traffic-flow conservation law. Equation (6.7) shows that the traffic flowing into a node equals the traffic flowing out of the node except the source node p and destination node q. Equation (6.8) shows that the flow out of the source node p is 1. The traffic-flow conservation law at the destination node q is guaranteed when Eqs. (6.7)–(6.8) are satisfied, which

is proved in [79]. Equation (6.9) shows the link-capacity constraints. This work imposes the upper bound $w_{ij}^{L} U_{t}^{L}$, which is stronger than w_{ij}^{L}. Equations (6.10)–(6.11) show the range of variables. The term m_{t}^{pq} in Eq. (6.9) can be formulated as follows:

$$m_{t}^{pq} = \sum_{k \in \mathcal{K}} B_{k}^{\mathrm{up}} \, \mathbb{I}_{k,t} \, z_{kp} \, y_{kq} \tag{6.12}$$

$$m_{t}^{qp} = \sum_{k \in \mathcal{K}} B_{k}^{\mathrm{down}} \, \mathbb{I}_{k,t} \, z_{kp} \, y_{kq} \tag{6.13}$$

$$(\forall p \in \boldsymbol{E}, \forall q \in \boldsymbol{N}, p \neq q).$$

Equation (6.12) shows the upload traffic demands from origin node p to destination node q. Here, z_{kp} and y_{kq} determine the node p and node q, and $\mathbb{I}_{k,t}$ extracts the executing tasks. Equation (6.13) shows the download traffic demands from node q to node p, which is the opposite of the upload.

Latency constraints

Task latency is the sum of all possible delays a task experiences during offloading, including processing, transmission, propagation, and queuing latency. Processing latency is the delay it takes to process the task at the edge or cloud server. This work refers to this latency as node latency τ_{k}^{N}. Transmission latency is the time it takes to push all the traffic onto the link, calculated by dividing the number of bits by the transfer rate. Propagation latency is the time it takes for traffic to travel from the origin to the destination, determined by the distance between nodes and the speed of signal propagation. For a wired network, propagation latency can be assumed to depend only on distance because the speed of light is constant in fiber. Queuing latency is the time that traffic waits in the queue before it can be processed. This work considers the sufficient buffer and assumes that the queuing latency is zero. This work refers to the sum of the transmission latency and the propagation latency as RTT latency τ_{k}^{RTT}.

Latency constraints of k^{th} task are formulated as follows:

$$\text{s.t.} \quad : \quad \tau_{k}^{N} + \tau_{k}^{\mathrm{RTT}} \leq \tau_{k}^{\max} \qquad (\forall k \in \mathcal{K}). \tag{6.14}$$

The definitions of the node latency τ_k^N and RTT latency τ_k^{RTT} of the k^{th} task are as follows:

$$\tau_k^N := \sum_{n \in \boldsymbol{N}} y_{kn} \frac{C_k}{v_n^N} \tag{6.15}$$

$$\tau_k^{\mathrm{RTT}} := \max_{p \in \mathcal{P}_k^{\mathrm{up}}} \left(\tau_p(B_k^{\mathrm{up}}) \right) + \max_{p \in \mathcal{P}_k^{\mathrm{down}}} \left(\tau_p(B_k^{\mathrm{up}}) \right). \tag{6.16}$$

Equation (6.15) shows that τ_k^N is determined by the computing-demand size of the task C_k and the node-processing capability per second v_n^N. The term C_k / v_n^N is the time it takes to process C_k at node n. The term y_{kn} determines the node n to which the task k is allocated, as explained in Eq. (6.15). Equation (6.16) shows that τ_k^{RTT} is determined by the bottleneck path with the maximum latency when the task goes through multiple paths. Here $\mathcal{P}_k^{\mathrm{up}}$ and $\mathcal{P}_k^{\mathrm{down}}$ are the set of upload and download paths of the k^{th} task, which is calculated on the basis of $\boldsymbol{X}_t$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$. The term $\tau_p(b)$ is the function that shows the latency when the traffic demand b goes through the path p. The $\tau_p(b)$ is formulated as follows:

$$\tau_p(b) := r_p \cdot \frac{b}{\min_{(i,j) \in \mathcal{L}_p} (w_{ij}^L)} + \sum_{(i,j) \in \mathcal{L}_p} \alpha_{ij}^L. \tag{6.17}$$

The first term in Eq. (6.17) shows the transmission latency when the traffic demand b goes through the path p. The term r_p is the traffic-splitting ratio of path p calculated by $\boldsymbol{X}_t$ that satisfies the following constraints: $\sum_p (r_p) = 1$. The term $\mathcal{L}_p$ is the link set along with path p. The term $\min_{(i,j) \in \mathcal{L}_p} (w_{ij}^L)$ indicates the bottleneck bandwidth on path p. The second term in Eq. (6.17) shows the propagation latency of path p, which is the sum of the propagation latency α_{ij}^L of the link $(i, j) \in \mathcal{L}_p$.

6.2 Proposed method

6.2.1 Overview

This work gives an overview of the proposed task-offloading method, which is based on Coop-MADRL. This work aims to find the optimal task offloading that minimizes Eq. (6.1) while satisfying the constraints in Eqs. (6.2)–(6.14).

The decision variables are the task-allocation variables $\boldsymbol{Y}$ and routing variables $\boldsymbol{X}_t$. The proposed method consists of two parts: Coop-MADRL and mathematical optimization. Coop-MADRL is responsible for finding the optimal $\boldsymbol{Y}$, and mathematical optimization is responsible for finding the optimal $\boldsymbol{X}_t$. Another possible way to find the solution of two variables is to solve them in sequence, such as finding the $\boldsymbol{Y}$ and then finding the $\boldsymbol{X}_t$. However, in this case, the $\boldsymbol{Y}$ is determined on the basis of a non-optimal $\boldsymbol{X}_t$, which may degrade the performance of the solution compared with that of the proposed method. Therefore, the proposed method jointly optimizes $\boldsymbol{Y}$ and $\boldsymbol{X}_t$. Concretely, this work calculates the reward of Coop-MADRL to find the optimal $\boldsymbol{Y}$ on the basis of the $\boldsymbol{X}_t$ calculated by mathematical optimization. The concept is based on current methods [56, 75, 76]. The proposed method also consists of centralized training and decentralized execution. The decentralized agents continually execute the task offloading after centralized training.

Table 6.4 summarizes the notation definitions of the proposed method. This work introduces $|\boldsymbol{E}|$ agents equal to the number of edges and assigns each agent for each edge offloading control. The e^{th} agent $g_e \in \mathcal{G}$ learns how to optimize task offloading for the e^{th} edge. In centralized training, each task arrives at the nearest edge at the beginning of each t. Each agent g_e observes the information o_t^e. On the basis of the observation, the g_e determines the node to offload the task as an action a_t^e. Each agent randomly determines offloading nodes in the early stages of training but can select the best node as training progresses. The proposed method then aggregates the traffic demands between nodes, then the mathematical-optimization solver calculates the optimal route between nodes. The solver calculates a routing variable $\boldsymbol{X}_t$ to minimize the link utilization U_t^L while satisfying the constraints in Eqs. (6.7)–(6.11). Since the routing variable $\boldsymbol{X}_t$ is a continuous value within 0–1, as shown in Eq. (6.11), this problem is classified as a linear programming problem. Next, each edge forwards tasks to the offloading nodes through the optimal route, and the proposed method calculates the reward r_t. By repeating these steps, agents collect learning samples that are combinations of $\langle \boldsymbol{o}_t, \boldsymbol{a}_t, r_t \rangle$. On the basis of these samples, the agents are trained using Coop-MADRL. In decentralized execution, the trained agents repeat the above steps except for the learning steps. Since each agent has already learned the cooperative action for all agents in centralized

Table 6.4: Notation definitions for proposed method.

Notation	**Definition**
$t^{\text{sim}} \in T^{\text{sim}}$	Time-step for simulator (T^{sim}: Total time steps)
$\mathcal{G} := \{g_e\}$	Agent set ($1 \leq e \leq \lvert \boldsymbol{E} \rvert$)
$s_t \in \mathcal{S}$	State at step t ($\mathcal{S}$: State space)
$\mathcal{O} := \{\mathcal{O}^e\}$	Observation sets for all agents
$o_t^e \in \mathcal{O}^e$	Observation for agent g_e at step t
$\boldsymbol{o}_t := \{o_t^e\}$	All observations at step t
$\mathcal{A} := \{\mathcal{A}^e\}$	Action sets for all agents ($\mathcal{A}^e$: Action space)
$a_t^e \in \mathcal{A}^e$	Action for agent g_e at step t
$\boldsymbol{a}_t := \{a_t^e\}$	All actions at step t
r_t	Reward for agent g_e at step t
$Q_e\left(o_t^e, a_t^n\right)$	Action-value function for agent g_e
$Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$	Joint action-value function for all agent
$\mathcal{M}$	Replay memory
$h^i \in \boldsymbol{h}$	Observation-action history
$\boldsymbol{h}$	Observation-action history of all agents

training, each agent can act independently in decentralized execution.

6.2.2 Modeling

This work first defines the variables that represent a subset of tasks as follows:

$$\mathcal{K}_t := \{k \in \mathcal{K} \mid \mathbb{I}_{k,t} = 1\} \tag{6.18}$$

$$\mathcal{K}_e := \{k \in \mathcal{K} \mid z_{ke} = 1\} \tag{6.19}$$

$$\mathcal{D}_t := \{\boldsymbol{D}_k \in \mathcal{D} \mid t_k = t\} \tag{6.20}$$

$$\mathcal{D}_{e,t} := \{\boldsymbol{D}_k \in \mathcal{D} \mid t_k = t, k \in \mathcal{K}_e\}. \tag{6.21}$$

The $\mathcal{K}_t$ indicates the index subset of tasks executed at time step t. The $\mathcal{K}_e$ indicates the index subset of tasks accepted at edge e. The $\mathcal{D}_t$ indicates the subset of tasks accepted at t. The $\mathcal{D}_{e,t}$ indicates the subset of tasks accepted on edge e at t, i.e., $\mathcal{D}_{e,t} \subset \mathcal{D}_t \subset \mathcal{D}$.

A state is defined as $s_t = [\mathcal{D}_t, \boldsymbol{U}_t^L, \boldsymbol{U}_t^S]$. An observation for agent g_e is defined as $o_t^e = [\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^N]$. The candidate action set $\mathcal{A}^e$ is defined as the set

of nodes that offload tasks. The node set consists of the nearest edge, neighboring edges, and neighboring clouds. The neighboring edges and clouds are one or more nodes determined by the physical-network graph $G(\boldsymbol{N}, \boldsymbol{L})$. This work excludes nodes from $\mathcal{A}^e$ that have no remaining resources and edges that are more than two hops away from edge e. When edge e does not receive any tasks at t, agent g_e chooses the action "do nothing." This work designs the reward function to return a negative value if the constraints are not satisfied; otherwise, a positive value depends on the objective-function value (see Section 6.2.5 for details).

6.2.3 Formulation

Algorithm 11 shows the centralized training using Coop-MADRL. Line 1 shows the initialization of agent parameters. A series of procedures (lines 2–18) is repeatedly executed until learning is complete. Lines 3–4 show the generation of tasks and initialization of environment parameters. A series of actions is called an episode, and each episode (lines 5–16) is repeatedly executed. In each episode, agents collect learning samples that are combinations of $\langle \boldsymbol{o}_t, \boldsymbol{a}_t, r_t \rangle$. This work denotes a time step for the network simulator as $t^{\text{sim}} \in T^{\text{sim}}$, which is reset at the beginning of each episode. In line 7, when edge e accepts multiple tasks at t^{sim}, agent g_e selects one task in an FIFO manner. The two task subsets can be written as the following relationship: $\mathcal{D}_{e,t} \subseteq \mathcal{D}_{e,t^{\text{sim}}} (\subset \mathcal{D})$. In line 9, a random action is selected with probability ε; otherwise, an action that maximizes $Q_e\left(o_t^e, a'\right)$ is selected with probability $1-\varepsilon$. This is to avoid convergence to a local optimum solution. Each agent executes lines 7–9 in parallel. In line 10, task offloading is updated in accordance with $\boldsymbol{a}_t$ by Alg. 13. Line 11 shows the reward calculation. Lines 12–13 mean the termination condition of agent learning. In this algorithm, $r_t \leq -1$ is the terminate condition, i.e., the state that does not satisfy at least one constraint. In lines 14–15, if all tasks accepted at t^{sim} are allocated, it proceeds to the next $t^{\text{sim}} + 1$. Line 17 shows stores in replay memory $\mathcal{M}$. The reason for storing the samples once in replay memory is to eliminate the time dependence of collecting training samples [9]. Lines 3–17 can be paralleled because the samples of the episodic transitions stored in replay memory are independent of the storing order. In

Algorithm 11 Centralized training of Coop-MADRL.

1: **initialize:** agent parameters, $t \leftarrow 0$
2: **while** $t \leq T$ **do**
3: generate tasks $\mathcal{D}$ for training
4: **initialize:** environment parameters
5: **for** $t^{\text{sim}} = 0, T^{\text{sim}}$ **do**
6: **for each** $g_e \in \mathcal{G}$ **do**
7: $\mathcal{D}_{e,t} \leftarrow$ select one task $(\mathcal{D}_{e,t^{\text{sim}}})$
8: $o_t^e \leftarrow$ observation $(\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^N)$
9: $a_t^e \leftarrow$ select epsilon greedy action $\left(o_t^e\right)$
10: update environment $(\boldsymbol{o}_t, \boldsymbol{a}_t)$ by Alg. 13
11: $r_t \leftarrow$ calculate reward by Alg. 14
12: **if** $r_t \leq -1$ **then**
13: terminate episode: $t^{\text{sim}} \leftarrow T^{\text{sim}}$
14: **if** all $\mathcal{D}_{t^{\text{sim}}}$ are allocated **then**
15: $t^{\text{sim}} \leftarrow t^{\text{sim}} + 1$
16: $t \leftarrow t + 1$
17: store episodic transition $\left(\boldsymbol{o}_j, \boldsymbol{a}_j, r_j\right), \forall j \in$ episode steps
18: train all agents $\mathcal{G}$ by random episodic transition

line 18, all agents $\mathcal{G}$ are trained by the history of episodic transition, which is randomly taken from $\mathcal{M}$. This work used VDN [54] as the learning algorithm for all agents (see Section 2.1.2 for details). To achieve global optimization, all agents learn to maximize a shared objective function, namely the joint action-value function $Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$, which helps prevent competition among the agents. In the proposed architecture, each edge has a replay memory $\mathcal{M}$. Since the observation o_t^e of the edge e is composed of local task information and shared network information, episodic transitions can be stored at each edge without aggregating data from other agents.

Algorithm 12 shows the Coop-MADRL algorithm of the proposed task-offloading method. Line 1 shows the pre-training of $\mathcal{G}$ by using Alg. 11. Next, this algorithm continually repeats lines 2–9 as long as the proposed method accepts new tasks. In line 6, each agent selects an a_t^e that maximizes $Q_e\left(o_t^e, a'\right)$.

Algorithm 12 Decentralized execution of Coop-MADRL.

1: $t^{\mathrm{sim}} \leftarrow 0$, $Q_e\left(o^e, a^e\right) \leftarrow$ train all agents $\mathcal{G}$ by Alg. 11
2: **while** all tasks are allocated **do**
3: **for each** $g_e \in \mathcal{G}$ **do**
4: $\mathcal{D}_{e,t} \leftarrow$ select one task $\left(\mathcal{D}_{e,t^{\mathrm{sim}}}\right)$
5: $o_t^e \leftarrow$ observation $\left(\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^N\right)$
6: $a_t^e \leftarrow \arg\max_{a' \in \mathcal{A}^e} Q_e\left(o_t^e, a'\right)$
7: update environment $\left(\boldsymbol{o}_t, \boldsymbol{a}_t\right)$ by Alg. 13
8: **if** all $\mathcal{D}_{t^{\mathrm{sim}}}$ are allocated **then**
9: $t^{\mathrm{sim}} \leftarrow t^{\mathrm{sim}} + 1$

Algorithm 13 Update environment.

1: $\boldsymbol{Y} \leftarrow$ allocate tasks $\left(\boldsymbol{a}_t, \mathcal{D}_t\right)$
2: $\boldsymbol{U}_t^N \leftarrow$ calculate node utilization $\left(\mathcal{D}_{\mathrm{t}}, \boldsymbol{Y}\right)$
3: $\boldsymbol{M}_t \leftarrow$ calculate traffic matrix $\left(\mathcal{D}_t, \boldsymbol{Y}, \boldsymbol{Z}\right)$
4: $\boldsymbol{U}_t^L, \boldsymbol{X}_t \leftarrow$ calculate link utilization $\left(\boldsymbol{M}_{\mathrm{t}}\right)$
5: $\tau_k^N, \tau_k^{\mathrm{RTT}} \leftarrow$ calculate latency $\left(\boldsymbol{X}_{\mathrm{t}}, \boldsymbol{Y}, \boldsymbol{Z}\right)$
6: **return** $\boldsymbol{U}_t^N, \boldsymbol{U}_t^L, \tau_k^N, \tau_k^{\mathrm{RTT}}$

Algorithm 14 Reward calculation.

1: $r_t \leftarrow \mathrm{Eff}\left(U_{t+1}^L\right) + \mathrm{Eff}\left(U_{t+1}^N\right) + \mathrm{Eff}\left(\tau_t^{\mathrm{ave}}\right)$
2: **if** $U_{t+1}^L > 1$ or $U_{t+1}^L > 1$ **then**
3: **return** -3
4: **return** $\max(-3, \min(3, r_t))$

6.2.4 Update environment

Algorithm 13 shows the procedure of the update environment. The task-allocation variables $\boldsymbol{Y}$ and routing variables $\boldsymbol{X}_t$ are updated in Alg. 13. Line 1 shows the calculation of $\boldsymbol{Y}$, line 2 the calculation of $\boldsymbol{U}_t^N$, line 3 the calculation of $\boldsymbol{M}_t$ using Eqs. (6.12)–(6.13), and line 4 the calculation of $\boldsymbol{U}_t^N$. In line 4, this work solves the route-optimization problem, which calculates $\boldsymbol{X}_t$ using Eqs. (6.7)–(6.11). Line 5 shows the calculation of latency. Finally, Alg. 13 returns variables for the reward calculation.

6.2.5 Reward calculation

This work designs the reward function on the basis of the objective function in Eq. (6.1). Algorithm 14 shows the procedure of the reward calculation for $\mathcal{G}$. The term $\mathrm{Eff}(x)$ shows the efficiency function and is defined as follows:

$$\mathrm{Eff}(x) = \begin{cases} -x+1 & (x \leq 1) \\ -2x & (1 < x). \end{cases} \tag{6.22}$$

This work designs this function so that the efficiency decreases as x increases. It returns a positive value depending on x if $x < 1$; otherwise, it returns a negative value. The decrease in efficiency doubles when $x > 1$. Note that this design is essentially independent of the effectiveness of the proposed method. The term τ_t^{ave} indicates the average task latency efficiency at t, which shows the average satisfaction of latency and is defined as follows:

$$\tau_t^{\mathrm{ave}} = \sum_{k \in \mathcal{K}_t} \left(\frac{\tau_k^N + \tau_k^{\mathrm{RTT}}}{\tau_k^{\max}} \right) / |\mathcal{K}_t|. \tag{6.23}$$

The terms τ_k^N and τ_k^{RTT} are calculated using Eqs. (6.15)–(6.16). If it does not satisfy $U_{t+1}^L > 1$ or $U_{t+1}^L > 1$, it returns -3. Finally, it returns a clipped reward within $-3 \leq r_t \leq 3$. The reason for clipping the reward is to increase the stability of agent learning [9].

6.3 Evaluation

This work evaluated the effectiveness of the proposed method through simulations in terms of performance, network topology dependency, computation time, and scalability regarding the number of tasks. This work also evaluated its generalization performance for unknown task patterns. This work prepared five comparison methods and four network topologies for the evaluation. For each agent's DNN layer, this work introduced DRQN [45]. This work used a three-layer DNN consisting of two fully connected layers and the GRU layer [80]. This work used Adam [90] to optimize all DNNs and set the number of neurons in the hidden layers to 64 for all DNNs. This work used Double-DQN [31] as the DRL algorithm. This work implemented the

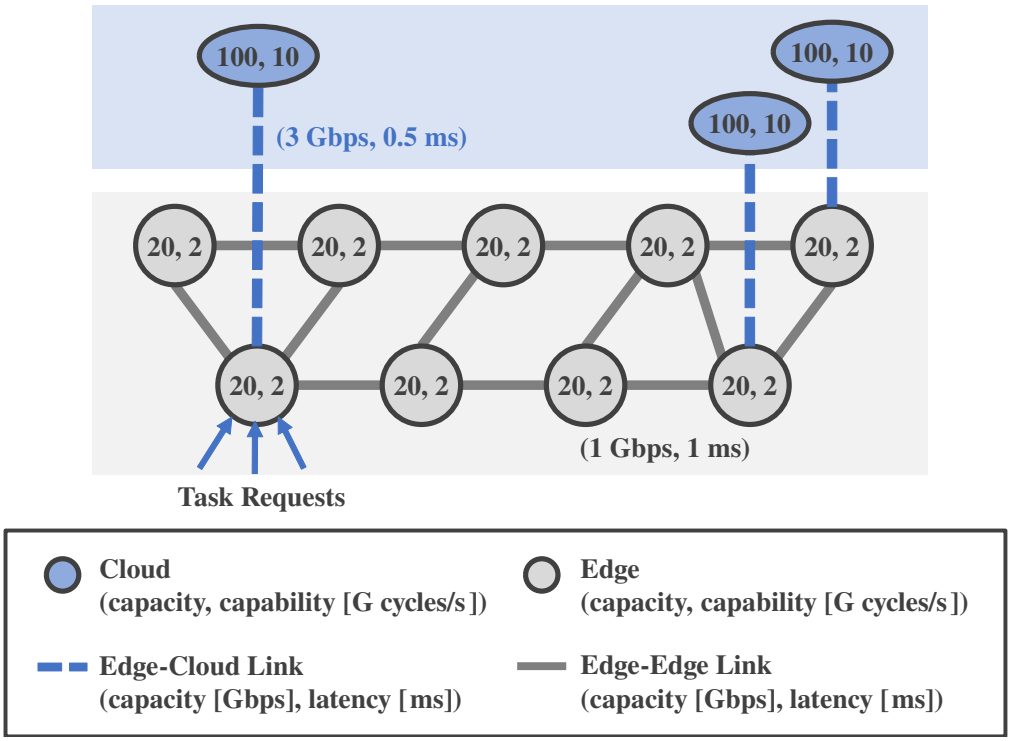


Figure 6.1: Network topology (Internet2). (©2023 IEEE.)

DRL-algorithm-based PyTorch [81], PyMARL [82], and PyMARL2 [91]. This work solved the route optimization using the GNU Linear Programming Kit (GLPK) [65]. For the hyperparameters of DRL, the learning rate was set to $\alpha = 0.001$, and the discount factor was set to $\gamma = 0.99$. The parameter ε, which determines the probability of random actions, was linearly decreased from 1.0 to 0.05 over the first 10^5 steps and then fixed at 0.05. This work accelerated the centralized training with parallel counts of 8.

Table 6.5: Task-generation parameters.

Parameter	**Type 1: Basic Task**	**Type 2: Download-heavy Task**	**Type 3: Computing-heavy Task**	**Type 4: Latency-sensitive Task**
B_k^{up} [Mbits]	0.1–50	0.1–1	90–100	0.1–10
B_k^{down} [Mbits]	0.1–50	90–100	0.1–1	0.1–10
C_k [G cycles]	0–1	0–1	10–20	0–0.1
$\tau_k^{\max}$ [ms]	$5(B_k^{\text{up}} + B_k^{\text{down}}) + 500C_k$ 1–1000	$5(B_k^{\text{up}} + B_k^{\text{down}}) + 500C_k$ 450.5–1005	$5(B_k^{\text{up}} + B_k^{\text{down}}) + 250C_k$ 2950.5–5505	$2.5(B_k^{\text{up}} + B_k^{\text{down}}) + 250C_k$ 0.5–75
Ratio	40%	20%	20%	20%

Table 6.6: Network-topology parameters.

Parameter	**Notation**	**Internet2**	**Abilene**	**Atlanta**	**Geant**
# of Edges	$\lvert \boldsymbol{N} \rvert$	9	12	15	22
# of Clouds	$\lvert \boldsymbol{C} \rvert$	3	3	3	3
# of Links	$\lvert \boldsymbol{L} \rvert$	16	18	25	39
Edge-processing capability	v_i^E [G cycles/s]	2	2	2	2
Cloud-processing capability	v_i^C [G cycles/s]	10	10	10	10
Edge capacity	w_i^E	20	20	20	20
Cloud capacity	w_i^C	100	100	100	100
Edge–Edge link capacity	w_{ij}^L [Gbps]	1	1	1	1
Edge–Cloud link capacity	w_{ij}^L [Gbps]	3	3	3	3
Edge–Edge propagation latency	α_{ij}^L [ms]	1	1	1	1
Edge–Cloud propagation latency	α_{ij}^L [ms]	0.5	0.5	0.5	0.5

6.3.1 Evaluation conditions

This work sets the number of tasks to $\mathcal{K} = 1000$, total time steps to $T = 5.0 \times 10^5$, total time steps of each episode to $T^{\mathrm{sim}} = 50$, and weighting parameter to $\lambda = 1$. This work assumes that each ED has one task during $t^{\mathrm{sim}} \in [1, T^{\mathrm{sim}}]$, the acceptance time t_k is randomly generated within 1–50, and the ED placement $\boldsymbol{Z}$ is randomly generated. This work sets the time-step interval from t to $t+1$ on the network simulator to 100 ms. Since this interval is merely a parameter of the network simulator, the actual training time per step need not be less than 100 ms. However, the execution time of real-world task offloading should be less than 100 ms. The proposed method satisfies these time constraints (see Section 6.3.4 for details).

For task generation, this work prepares four types of tasks assuming various use cases. This work uses a generalized task model to reveal that the proposed method can effectively allocate tasks even when many types of tasks are mixed. Table 6.5 summarizes the parameters for task generation. The task model is parameterized by upload and download traffic demand, B_k^{up} and B_k^{down}, computation demand C_k, and maximum permissible latency to accomplish the task $\tau_k^{\max}$. This work determines the maximum permissible latency on the basis of the upload and download traffic and computation demand. This work models Type 1 tasks as basic tasks and models other types of tasks by modifying the parameters of basic tasks. Each task parameter randomly takes a continuous value within a specified range. The task parameters are reset at the beginning of each episode.

For the physical network, this work prepares four network topologies. This work first prepares the 12-node topology on the basis of Internet2 [64], which consists of nine edges and three clouds, as shown in Fig. 6.1. This work connects each cloud to one randomly selected edge and fixes the cloud placements for all evaluations. The values in this figure show the pair of node-processing capability and node capacity (w_i^N, v_i^N) and the pair of link capacity and propagation latency $(w_{ij}^L, \alpha_{ij}^L)$. All evaluations except for the network-topology dependency evaluation were based on this topology. This work then prepares the other topologies that refer to SNDLib [84], a library of test instances of survivable fixed telecommunication network design. Table 6.6 summarizes the

Table 6.7: Summary of the proposed method and comparison methods.

Method	**Methodology**	**RL-based**	**Cooperation**
VDN (Ours)	Coop-MADRL	✓	✓
IQL	MADRL	✓	-
RA	Random Algorithm	-	-
GA	Greedy Algorithm	-	-
HA	Heuristic Algorithm	-	-
ES	Exhaustive Search	-	-

network-topology parameters. For each network topology, this work defines neighboring edges and clouds for each edge. This work defines the edges with a hop count of one as the neighboring edges for each edge. This work also defines the cloud(s) other than the farthest cloud from each edge as neighboring clouds for each edge.

This work performs training and evaluation steps on the prepared tasks and physical networks. This work runs Algorithm 11 for training and Algorithm 12 for evaluation. Tasks in training and evaluation are generated under the same conditions. Note that the tasks generated during training and evaluation are different because some parameters are randomly generated. In the evaluation, the MADRL-based methods select the best action (i.e., the offloading server) on the basis of the trained agents in line 6 of Algorithm 12. The non-DRL-based methods select the offloading server on the basis of their respective algorithms, rather than selecting actions by agents.

6.3.2 Comparison methods

Table 6.7 summarizes the proposed method and the comparison methods. VDN indicates the proposed Coop-MADRL-based method. IQL indicates the MADRL-based method without cooperation, i.e., each agent learns on the basis of their reward. As mentioned in Section 2.4, there is no method for considering multi-cloud networks and network topology, but IQL corresponds to such a method [35] when it considers them. Note that this work did not evaluate single-agent DRL because learning will clearly be unsuccessful due to requirements for huge training iterations until the Q-values for all actions are

sufficiently close to the optimal.

This work also compared the proposed method with three algorithms that may be able to offload tasks in a computation time close to RL-based methods. The random algorithm (RA) has a policy that allocates each task to a node randomly chosen from all candidate nodes at each t. The greedy algorithm (GA) has a policy that allocates tasks to the nearest edge until the node utilization of the nearest edge exceeds a threshold and allocates tasks to the neighboring cloud after exceeding that threshold. This work selected the best threshold with the best performance after evaluating all patterns in increments of 0.1 through a preliminary experiment. The GA threshold was set to 0.2, 0.2, 0.2, and 0.1 for Internet2, Abilene, Atlanta, and Geant, respectively. The heuristic algorithm (HA) has a policy that allocates tasks to suitable nodes depending on the characteristics of the task type. For this evaluation, HA allocated tasks to random nodes when the task type was 1 or 2, to the neighboring clouds when the task type was 3, and to the nearest edge when the task type was 4. This is based on the empirical rule that allocating computing-heavy tasks to the clouds and latency-sensitive tasks to the nearest edge are suitable. Note that these three algorithms also optimize routes calculated with the route-optimization algorithm at each t.

This work also evaluated exhaustive search (ES), which has a policy that finds the best offloading node by exhaustively calculating each reward for all candidate nodes at t, equivalent to solving an online optimization problem every t. Note that the solution of ES is different from the exact optimal solution. Whereas the optimal solution is defined as the solution that maximizes the sum of the rewards at each time, as shown in Eq. (6.1), ES aims to maximize the immediate reward obtained in the present. ES also independently calculates the optimal offload node for each edge at t on the basis of the information obtained at $t-1$, which may concentrate the tasks on certain nodes.

6.3.3 Evaluation results

6.3.4 Training curve

Figure 6.2 shows the training curves tracking the agent's total return of VDN and IQL under the Internet2 topology. The total return is defined as the sum

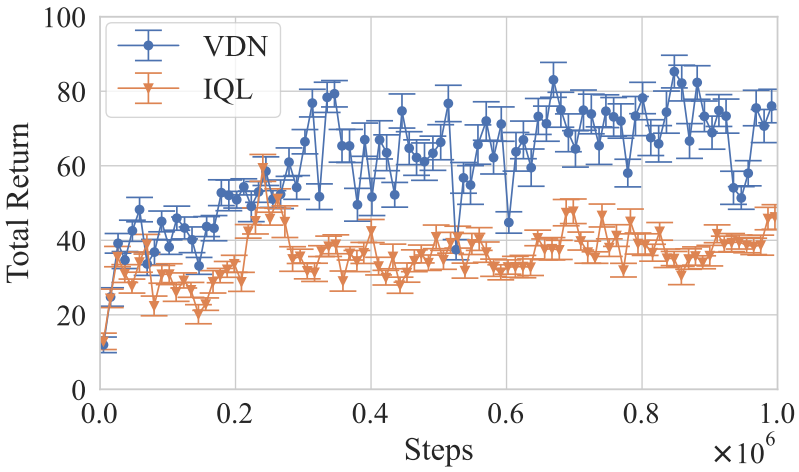


Figure 6.2: Training curves tracking agent's total return in Internet2 topology. (©2023 IEEE.)

of rewards at each time until the end of the episode. For this evaluation, the total time step was set to $T = 1.0 \times 10^6$ to ensure that the agent's learning has converged sufficiently. This work conducted five trials for every 1.0×10^4 steps with random initial conditions. The width of each bar indicates the standard deviation ($\pm\sigma$).

The results indicate that the average total return of the two MADRL methods increased as the training progress and converged around $T = 3.0 \times 10^5$. The results also indicate that the final return of VDN was higher than IQL, which means that IQL cannot learn the suitable task offloading that maximizes the objective function while satisfying constraints. This work discusses the performance details in Section 6.3.4.

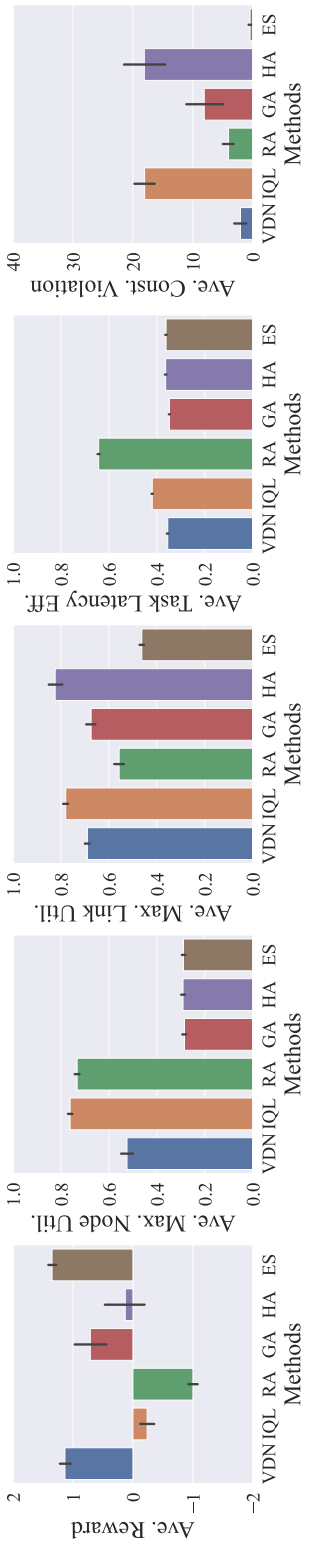


Figure 6.3: Performance in Internet2 topology. (©2023 IEEE.)

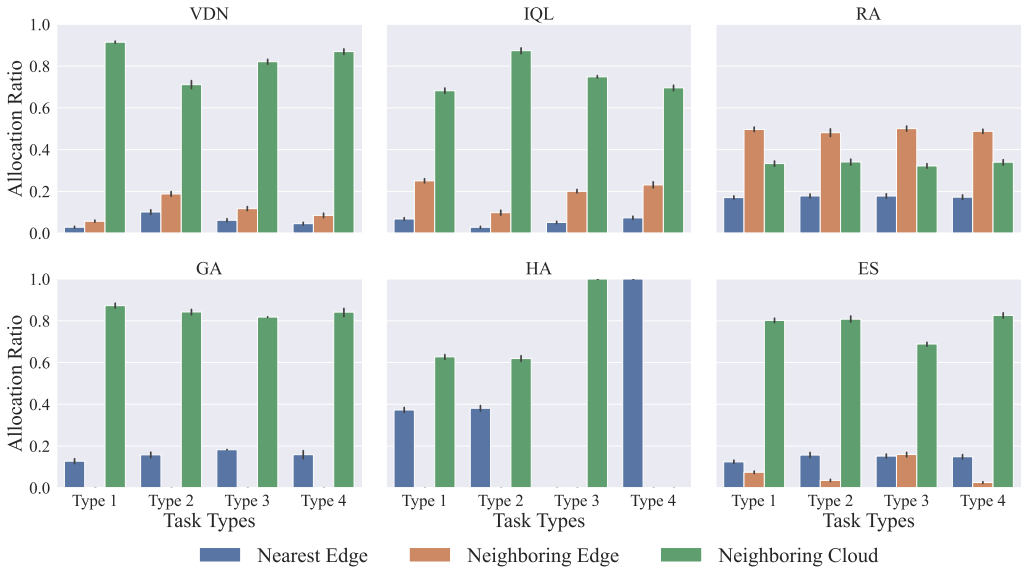


Figure 6.4: Allocation ratio of each task type in Internet2 topology. (©2023 IEEE.)

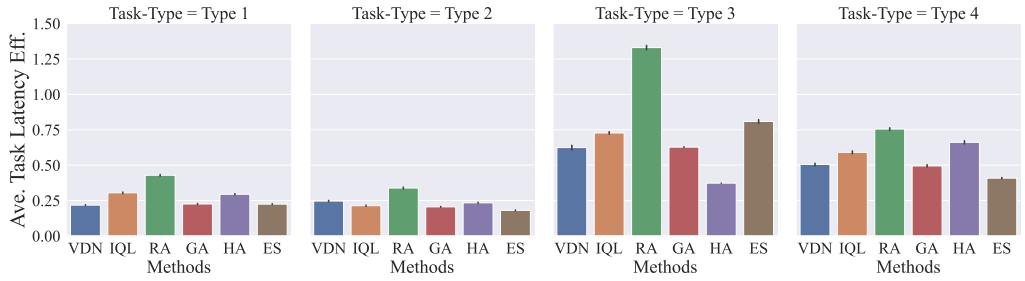


Figure 6.5: Task latency of each task type in Internet2 topology. (©2023 IEEE.)

Performance

Figure 6.3 shows the average performance of each method. This work carried out 20 calculations with random initial conditions and set the same random seeds for all trials. The width of each bar indicates the standard deviation ($\pm\sigma$). This work can roughly compare the performance of each method with the average reward. This work also investigated the performance details in terms of the second to fifth metrics in Fig. 6.3. These metrics are the average maximum node utilization U_t^N, average maximum link utilization U_t^L, average

task-latency efficiency τ_t^{ave}, and average constraint violation. The higher reward means better performance, whereas the lower values for other indicators mean better performance. When all constraints are satisfied, the node utilization, link utilization, and task latency take a value within 0–1. The constraint violation denotes the total number of times each method violates either node utilization, link utilization, or task-latency constraints.

Figure 6.3 shows that the proposed method (VDN) performed comparably to ES and outperformed the other methods. These results indicate that VDN and ES can maximize network-utilization efficiency and minimize task latency while reducing constraint violations. Note that constraint violations occurred for all methods because this work evaluated performance under severe conditions that put heavy loads on the network. Although ES performed the best, it is problematic in terms of computation time (see Section 6.3.4 for details).

Next, this work discusses the performance comparison between VDN and other methods. This work first compared VDN and IQL. Figure 6.3 shows that VDN performed better than IQL for all metrics. This indicates that the coordination among agents with VDN improves the performance of task offloading and can prevent constraint violations. Since each agent of IQL acts independently, task offloading is concentrated on lightly loaded resources in some cases, causing constraint violations as described in Chapter 1.

This work next compared VDN and the three non-RL-based computationally lightweight methods: RA, GA, and HA. The GA threshold was set to 0.2 because the best performance was obtained in the preliminary experiment. Figure 6.3 shows that VDN outperformed the three methods in terms of average reward. However, VDN had a higher maximum node utilization than GA and HA and higher maximum link utilization than RA. The reason is that agents choose the action that maximizes the average reward even if the node and link utilization are increased. As a characteristic of each method, GA performed comparably to or better than VDN in terms of maximum node utilization, maximum link utilization, and task latency efficiency by setting the best threshold in the preliminary experiment. However, GA had more constraint violations because it could not dynamically select allocations on the basis of the situation, resulting in a lower average reward than VDN. In addition, the maximum node utilization and task latency efficiency of HA were

Table 6.8: Average reward of each method for various network topologies.

Method	**Internet2**	**Abilene**	**Atlanta**	**Geant**
VDN	1.13 ± 0.21	0.64 ± 0.39	1.23 ± 0.15	1.32 ± 0.18
RA	-1.00 ± 0.18	-0.79 ± 0.30	-0.53 ± 0.22	-0.34 ± 0.15
GA	0.71 ± 0.61	-0.83 ± 0.51	0.61 ± 0.49	0.89 ± 0.34
HA	0.13 ± 0.76	-1.83 ± 0.21	-0.15 ± 0.63	-0.06 ± 0.55
ES	1.35 ± 0.15	0.92 ± 0.28	1.35 ± 0.23	1.53 ± 0.18

competitive with those of the other methods because HA designs the appropriate offloading node depending on the characteristics of each task. However, the other indicators of HA showed the worst performance because HA does not take into account the link constraints.

This work discusses the reasons for the performance shown in Fig. 6.3 by analyzing the latency and allocation ratio of each task type. Figures 6.4 and 6.5 show the latency and allocation ratio of each task type in the evaluation in Fig. 6.3. Here, the allocation ratio indicates the proportion of where tasks allocated to the nearest edge, neighboring edges, or neighboring clouds. The allocation ratios for VDN, IQL, and ES are determined by the task and network conditions. In contrast, the allocation ratios for RA and HA are fixed, regardless of the task and network conditions, and are determined by the algorithm design. The allocation ratio for GA is determined by the GA threshold value, which was set to 0.2 in this evaluation.

Figure 6.4 shows that VDN, IQL, GA, and ES primarily allocated tasks to the neighboring cloud. The result shows that the neighboring cloud is the preferred node for allocating tasks. This is because of the sufficient cloud-processing capacity and the short transmission delay between the edge and cloud, making assigning tasks to the cloud preferable in this evaluation condition. However, this result is only an example of an evaluation and does not deny the effectiveness of EC. The allocation to the edge may become dominant depending on the evaluation conditions.

Figure 6.4 shows that VDN changed the allocation ratio in accordance with the task type. VDN mainly allocated all types of tasks to the neighboring cloud. Looking at the ratio using the edges, VDN allocated most

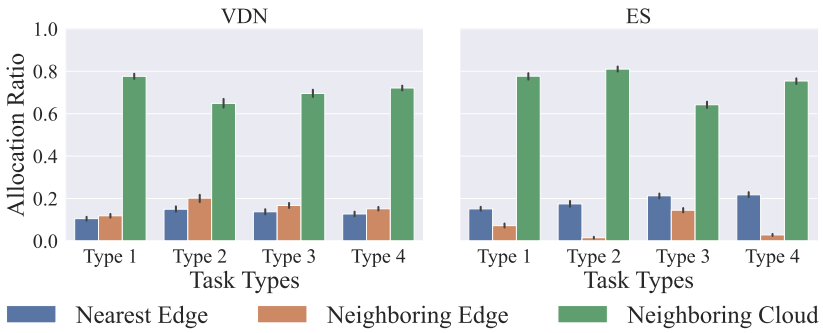


Figure 6.6: Allocation ratio of each task type in Abilene topology. (©2023 IEEE.)

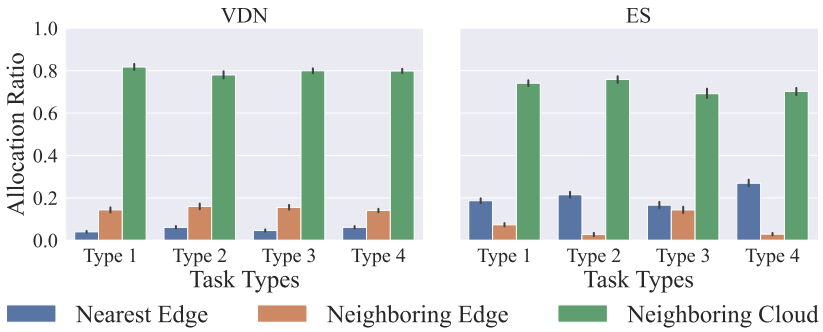


Figure 6.7: Allocation ratio of each task type in Atlanta topology. (©2023 IEEE.)

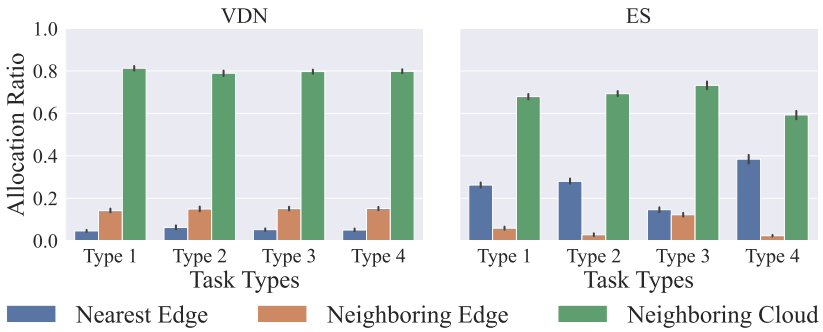


Figure 6.8: Allocation ratio of each task type in Geant topology. (©2023 IEEE.)

download-heavy tasks with high transmission costs (Task 2) to the edges. However, it allocated more computing-heavy tasks (Task 3) to neighboring clouds, even though the average traffic volumes of Tasks 2 and 3 were equal. This work concludes that VDN learns a superior policy through cooperative learning. However, the latency-sensitive tasks (Task 4) were primarily allocated to neighboring clouds, even though they should be allocated to the nearest edge to minimize latency. This work assumes the reason is that VDN attempts to maximize the overall network efficiency by allocating lightweight tasks to the cloud because of limited resources at the nearest edge.

Figure 6.4 shows that IQL changed the allocation ratio in accordance with the task type, although it is not associated with satisfactory performance, as shown in Figure 6.3. RA and GA did not change the allocation ratio in accordance with the task type, which is assumed to be one reason for the low performance of these methods. Figures 6.4 and 6.5 show that HA was suitable for latency for Tasks 3 and 4 because it changes the allocation node depending on the task type. However, as mentioned above, HA performed poorly because it does not take into account the link constraints.

Figure 6.4 shows that ES can use the nearest edges compared with VDN, which is assumed to be one reason that ES performed better than VDN. Even though ES selects the node that maximizes the reward for all candidate nodes, the performance difference between VDN and ES is slight. This is because ES aims to maximize the immediate reward obtained in the present. That is, selecting the node that maximizes the immediate reward in the present may not maximize the expectation of the sum of rewards obtained in the future.

Network-topology dependency

Table 6.8 shows the average reward of each method for various network topologies. This work carried out 20 calculations with random initial conditions and calculated the mean value and standard deviation of rewards. This work did not evaluate IQL because it is evident that learning agents is more difficult in larger topologies. The results indicate that VDN outperformed the other methods except for ES in all network topologies. Thus, the proposed method can be applied to larger network topologies.

This work first discusses the average performance with each topology. Table 6.8 shows that the average performance of all methods in Abilene decreased compared with that of other topologies. Comparing the conditions between Internet2 and other topologies, as shown in Table 6.6, only the network size is larger, and the other parameters are the same. This work considers that the graph structure of the topology determines the performance.

This work next discusses the performance of each method. Table 6.8 shows that VDN performed comparably to ES in all topologies, whereas the performance difference between VDN and ES increased in Abilene. Since all methods performed worse in Abilene, the performance difference between VDN and ES could be significant in environments where agent learning is complex. Table 6.8 also shows that the performance trends of GA and HA remain unchanged. On the other hand, the performance of RA improved as the topology size increased. This is because a larger topology provides more control options, enables load balancing, and improves performance regardless of the control method.

Figures 6.6–6.8 show the allocation ratio of each task type for larger topologies. This work omits the results of RA, GA, and HA because these methods do not change the allocation ratio depending on evaluation conditions. These figures show that VDN mainly allocated all types of tasks to the neighboring cloud, similar to that of Internet2. On the other hand, the allocation ratio of ES depended on network topology.

Figure 6.6 shows that the allocation ratio of VDN in Abilene was about the same as that in Internet2, while it increased the task allocation to the nearest edge. This means that VDN changed the allocation ratio in accordance with Abilene and learned the superior policy in accordance with the environment. Figures 6.7–6.8 show that the allocation ratio in larger topologies was similar to that of Internet2. However, the difference in the allocation ratio for each task type became smaller as the topology size increased. This means that learning in accordance with task type becomes more difficult as the topology size increases.

In summary, VDN outperformed the other methods except for ES in all network topologies. In addition, VDN can allocate tasks in accordance with the task type when the topology is small, but this becomes more difficult as the topology size increases. Although VDN cannot learn by task type in

Table 6.9: Execution times for various network topologies.

Method	**Internet2**	**Abilene**	**Atlanta**	**Geant**
VDN (Ours)	0.19 ms	0.26 ms	0.26 ms	0.25 ms
RA	0.07 ms	0.08 ms	0.09 ms	0.10 ms
GA	0.08 ms	0.12 ms	0.12 ms	0.11 ms
HA	0.11 ms	0.23 ms	0.28 ms	0.45 ms
ES	0.89 s	2.5 s	6.0 s	32.4 s

Table 6.10: Training times for various network topologies.

Method	**Internet2**	**Abilene**	**Atlanta**	**Geant**
VDN (Ours)	0.3 days	0.6 days	1.7 days	5.8 days

larger topologies, it achieves superior performance competitive with that of ES by learning the appropriate allocation to edges and clouds depending on the situation.

Computation time

Table 6.9 shows the average execution time of each method for various network topologies. The execution time denotes the computation time required to determine the allocated node for a single task, which corresponds to line 4 in Alg. 12 for VDN. This work used Apple M1 Ultra for the evaluation. The execution time of all methods except for ES was less than 1 ms for all topologies. The execution time tended to increase as topology size increases, but the increase is almost negligible for task latency. This means that VDN performs sufficiently in terms of computation time. However, the execution time of ES was about one second in the fastest case. The execution time of these task-offloading methods was added to the task latency in real-world scenarios. When the execution time takes one second, the task latency cannot be shorter than one second. Therefore, it is a critical for many tasks to keep latency to less than a few tens of milliseconds. The execution time drastically increased as the topology size increased because the computational complexity of the optimization calculation also drastically increased. This work concludes that VDN is the best task-offloading method, considering the allocation performance shown

in Table 6.8 and execution times shown in Table 6.9.

Table 6.10 shows the training time of VDN for various network topologies. The training time denotes the computation time required for the agents to complete the training of all time-steps $T = 5 \times 10^5$. The training time increases as topology size increases. Most of the increase in the training time was due to the increased computational complexity of the network simulation, which corresponds to Alg. 13. Although larger topologies require longer training times, this is not problematic since the training only needs to be done once.

Table 6.11: Latency-weighting-parameter dependency.

Case	**Ave. Max. Node Util.**	**Ave. Max. Link Util.**	**Ave. Task Latency Eff.**	**Constraint Violation**
$\lambda = 1$	0.52 ± 0.06	0.69 ± 0.02	0.35 ± 0.00	2.1 ± 2.5
$\lambda = 2$	0.55 ± 0.07	0.71 ± 0.02	0.34 ± 0.00	2.8 ± 2.4
$\lambda = 3$	0.34 ± 0.04	0.74 ± 0.03	0.33 ± 0.00	3.8 ± 3.2

Table 6.12: Network parameters for scalability evaluation.

Case	w_i^E	w_i^C	w_{ij}^L	w_{ij}^L
$\mathcal{K}$ = 1000	20	100	1	3
$\mathcal{K}$ = 2000	38	190	1.9	5.7
$\mathcal{K}$ = 5000	95	475	4.75	14.25
$\mathcal{K}$ = 10000	180	900	9	27

Table 6.13: Scalability evaluation regarding number of tasks.

Case	**Ave. Reward**
$\mathcal{K}$ = 1000	1.13 ± 0.21
$\mathcal{K}$ = 2000	1.36 ± 0.25
$\mathcal{K}$ = 5000	1.25 ± 0.60
$\mathcal{K}$ = 10000	1.19 ± 0.61

Latency-weighting-parameter dependency

Table 6.11 shows the weighting-parameter dependency of the objective function shown in Eq. (6.1). The weighting parameter λ determines the importance ratio of resource efficiency and task latency. This work carried out 20 calculations with random initial conditions and set the same random seeds for all calculations. The results indicate that, as λ increased, VDN improved in task-latency efficiency and worsened the sum of node and link utilization. This work revealed that the proposed method can adjust the importance of the objective function by λ. However, these indicators show no drastic change for the increase in λ. This is because reducing task latency leads to better node and link utilization. For example, to reduce task latency, the task-offloading method must balance server loads between clouds and edges in accordance with the task characteristics, which improves node utilization. To reduce task latency, it must minimize the route length of the task, which decreases link loads and improves link utilization.

Scalability regarding number of tasks

This work evaluated the scalability of VDN regarding the number of tasks $\mathcal{K}$. This work simultaneously evaluated the generalization performance of VDN regarding the number of tasks. The generalization performance measures how accurately a method can perform for previously unseen data. In other words, this work evaluated the performance of VDN with a different number of tasks than during training. This work increased link capacity and node capacity depending on $\mathcal{K}$, as shown in Table 6.12. Therefore, the relative load on the system is kept constant and this work can only evaluate the scalability of VDN. Other parameters were those in Table 6.6. This work evaluated performance under strict conditions where the capacity increase is 95% of the constant multiple.

Table 6.13 shows the scalability of VDN regarding the number of tasks $\mathcal{K}$ in Internet2. This work carried out 20 calculations with random initial conditions and calculated the average and the standard deviation of rewards. This work uses agents trained when $\mathcal{K} = 1000$ for all conditions. VDN kept the average rewards even as the number of tasks increased. The agents of VDN learn the policy from accepted task information and each node and link utilization information. Since this information of the physical network is normalized and independent of the number of tasks, VDN can determine the preferred task allocation without depending on the number of tasks. This work concludes that the proposed method has scalability and generalization performance regarding the number of tasks. However, the variance of rewards tends to increase as the number of tasks increases. Therefore, retraining the agents is advisable when the number of tasks significantly differs from the training condition.

Table 6.14: Various cases for generalization-performance evaluation.

Case	**Task 1**	**Task 2**	**Task 3**	**Task 4**
Default	40%	20%	20%	20%
Task2–30%	30%	30%		
Task2–40%	20%	40%	20%	20%
Task2–50%	10%	50%		
Task3–22%	38%		22%	
Task3–23%	37%	20%	23%	20%
Task3–25%	35%		25%	
Task4–30%	30%			30%
Task4–40%	20%	20%	20%	40%
Task4–50%	10%			50%
All–15%	55%	15%	15%	15%
All–25%	25%	25%	25%	25%
All–30%	10%	30%	30%	30%

Table 6.15: Generalization-performance evaluation for various task types.

Case	**Ave. Reward**	**Ave. Max. Node Util.**	**Ave. Max. Link Util.**	**Ave. Task-latency Eff.**	**Constraint Violation**
Default	1.13 ± 0.21	0.52 ± 0.06	0.69 ± 0.02	0.35 ± 0.00	2.1 ± 2.5
Task 2–30%	1.03 ± 0.26	0.54 ± 0.05	0.69 ± 0.03	0.36 ± 0.00	2.8 ± 2.9
Task 2–40%	1.03 ± 0.24	0.55 ± 0.08	0.70 ± 0.03	0.36 ± 0.00	2.6 ± 2.7
Task 2–50%	0.94 ± 0.37	0.56 ± 0.05	0.70 ± 0.04	0.36 ± 0.00	4.1 ± 5.0
Task 3–22%	0.70 ± 0.36	0.58 ± 0.03	0.75 ± 0.03	0.37 ± 0.00	6.1 ± 4.8
Task 3–23%	0.34 ± 0.60	0.59 ± 0.03	0.79 ± 0.07	0.37 ± 0.00	10.6 ± 7.7
Task 3–25%	−0.36 ± 0.69	0.61 ± 0.04	0.88 ± 0.07	0.38 ± 0.00	19.6 ± 9.0
Task 4–30%	1.08 ± 0.23	0.49 ± 0.05	0.70 ± 0.03	0.38 ± 0.00	2.9 ± 3.1
Task 4–40%	0.88 ± 0.28	0.48 ± 0.07	0.73 ± 0.04	0.42 ± 0.00	5.3 ± 3.8
Task 4–50%	0.60 ± 0.38	0.43 ± 0.04	0.76 ± 0.03	0.44 ± 0.00	9.9 ± 5.1
All–15%	1.57 ± 0.11	0.35 ± 0.05	0.61 ± 0.03	0.31 ± 0.01	0.3 ± 1.1
All–25%	−0.14 ± 0.53	0.62 ± 0.04	0.85 ± 0.04	0.39 ± 0.00	16.6 ± 7.3
All–30%	−1.28 ± 0.47	0.63 ± 0.04	1.03 ± 0.08	0.42 ± 0.00	32.4 ± 6.3

Generalization performance for task types

This work evaluates the generalization performance of VDN for various task types. This work evaluates the average reward when each task type ratio differed from the training condition. Table 6.14 shows various cases with different ratios of task types. This work sets the conditions in Table 6.5 as default. This work prepared cases that increase the ratio of each task type and decrease the ratio of basic tasks. This work also prepared cases that change the ratio of three task types per 5% and the ratio of basic tasks per 15%.

Table 6.15 shows the average reward and other indicators for various cases changing the ratio of task types. This work carried out 20 calculations with random initial conditions and set the same random seeds for all calculations. The average reward decreased as the ratio of Tasks 2–4 increased from the training condition. For Task 3, VDN had generalization performance until the task ratio increased by 2%. The average reward notably decreased for increases of 5%. This is because tasks with high average computing demand occupy the computing resources over a long period and reduce node utilization. For Tasks 2 and 4, VDN had generalization performance even when the task ratio increased by 20%. This is because Tasks 2 and 4 occupy computing resources for only a short time. Similarly, when this work simultaneously increased the ratio of the three task types by 5%, VDN showed no generalization performance. In other words, the proposed method has generalization performance when the prediction error in the proportion of computing-heavy tasks is within 2%. Conversely, for the case in which this work simultaneously decreased the ratio of the three task types, the performance of VDN improved. Therefore, this work concludes that the proposed method can have generalization performance for various task types by pre-training with the predicted maximum percentage of resource-consuming tasks.

6.3.5 Discussion

This work discusses the future work of the proposed method. This work has demonstrated the effectiveness of the proposed method. However, there are still challenges that this work should explore for applying the proposed method in a commercial environment or a future network.

This work discusses the challenge of the applicability of the proposed method. This work assumes that the routes in the access network take the shortest path. Current wired access networks with optical fiber automatically select the shortest path from a few routing options. Current wireless access networks for cellular automatically select the best BS for each ED on the basis of the received signal power. Therefore, the proposed method is applicable in real-world scenarios of the current network. However, there are some challenges in applying the proposed method in a commercial environment or a future network.

The first challenge is scalability for edges. The proposed method may only be applicable to networks with EC servers with dozens of edges. This work targets commercial networks that cover a large area, such as networks in multiple regions or parts of a country. Such an access network includes more than 10 000 edges. Similar to other methods [32–39] that can only handle a few dozen BSs at most, as described in Section 2.4, the proposed method also cannot control more than 10 000 edges. This work deploys each agent at each edge (i.e., at each BS), and each agent learns cooperative control. Therefore, the agents should learn cooperative control among many agents in the mentioned environment. However, this is difficult to achieve with current technology and is one of the future works. As a possible solution, the combined method of Mean-field Game (MFG) theory and DRL may handle a large-scale task-offloading system with many BSs. MFG theory [92] studies strategic decision-making by small interacting agents in large populations, inspired by mean-field theory in physics. Each agent minimizes or maximizes the problem objective, taking into account the decisions of the other agents. Several studies have addressed network control methods that combine MFG theory and DRL, such as unmanned aerial vehicle control [93, 94] and computation offloading in EC [95–97].

The next challenge is scalability for tasks. The processing time of the proposed method for determining the offload server is about 0.2 ms per task. Also, because the proposed method processes tasks sequentially, one at a time, it cannot handle thousands or millions of tasks per second required in real-world environments. Therefore, sufficient scalability for tasks is one of the future works. One possible solution is parallelization. Assigning many trained

agents at each edge makes it possible to process many tasks simultaneously. Many trained agents working independently may cause new problems.

The other challenge is to optimize task offloading for an entire end-to-end network. Since task latency can be expressed as the sum of backbone and access networks, this work can find near-optimal task offloading and routes between EDs and servers by optimizing the backbone and access networks independently. However, future networks will require more efficient optimal task offloading by considering the entire end-to-end multi-domain between EDs and servers, such as backbone network, access network, edge servers, and cloud servers. For this purpose, the proposed method should consider the latency and link quality of the wireless network between EDs and BSs. The difficulty in the wireless network is that the observed information is incomplete or uncertain. Several studies have addressed task offloading in the wireless network under imperfect channel state information (CSI) [98–103]. In particular, security is one of the major concerns for mobile EC with incomplete CSI. Several studies have been conducted on security-aware task allocation under incomplete CSI [104–106]. In addition, several studies have been conducted on end-to-end network slicing or multi-domain network slicing to integrate the multiple controls [107–109]. Although much research has been done, optimizing task offloading in an end-to-end network remains an unsolved problem and one of the major challenges. By combining the proposed method with above studies, the problem should be addressed in the future.

6.4 Chapter summary

This chapter formulated an optimal task-offloading problem and proposed a cooperative task-offloading method for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. This method is based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly achieve efficient task offloading by learning the relationship between network-demand patterns and optimal task offloading by using deep reinforcement learning in advance. This method also introduces a cooperative multi-agent technique, improving the efficiency of task offloading. Evaluations revealed that the proposed method can minimize network utiliza-

tion and task latency while minimizing constraint violations in less than 1 ms in various network topologies. They also revealed that cooperative learning improves the efficiency of task offloading. This work demonstrated that the proposed method can have generalization performance for various task types by pre-training with many resource-consuming tasks. This work plans to evaluate the performance of the proposed method and conduct further detailed analysis in more complicated use cases or real-world applications. This work also plans to improve the scalability and interpretability of the proposed method.

Chapter 7

Conclusions

Network functions virtualization (NFV) and edge computing (EC) are the key technologies of future networks. In NFV, a significant research challenge is to determine the optimal allocation of virtual networks (VNs) while taking into account the constraints of limited network and server resources. Similarly, in EC, a critical research challenge is to identify efficient strategies for offloading tasks to edge servers or cloud servers, taking into account the diverse characteristics of tasks and the constraints of network and server resources. Network resource management of future networks must be optimized to maximize resource utilization efficiency in a network with limited physical resources. The performance of allocation algorithms is critical for future network management, as it determines the overall network's resource utilization efficiency. This thesis focused on reinforcement learning (RL) as a solution to the resource management problem because it can quickly calculate a near-optimal resource allocation by learning the relationship between input network resource patterns and output resource allocation in advance. This thesis studied four specific problems about network resource management using multi-agent deep reinforcement learning (MADRL). Each problem corresponds to resource-integrated control in NFV, dynamic VN allocation, and task offloading for multi-cloud-edge networks, respectively.

Firstly, this thesis presented an extendable resource-integrated control method in NFV by coordinating multiple control algorithms. This work developed an efficient coordination algorithm on the basis of RL, which makes it possible to

find better solutions with fewer explorations by learning a strategy that can improve resource-utilization efficiency with each exploration step. Simulations revealed that the proposed algorithm can improve solution exploration for 12 representative types of the virtual network allocation use cases modeled from previous studies. This qualitatively revealed that the proposed method has extendability. This work also found that it can improve resource-utilization efficiency by 22% and total reliability by 8% in less than 5000 steps in the case of several hundred virtual machines (VMs) and a hundred intrusion detection systems (IDSs).

Secondly, this thesis proposed a dynamic virtual network allocation method based on the proposed Safe-MADRL. This method can quickly optimize the network resources even when traffic demand change drastically. It can also reduce the agent's constraint violations such as the control that leads to network congestion and server overload. Simulations revealed that the proposed method can reduce the maximum link utilization to half while maintaining the maximum server utilization compared to the static allocation method under practical-network conditions. Moreover, the computation time of the proposed method was less than one second, which is a significant increase in speed compared to exhaustive search. As a result, this work revealed that the proposed method simultaneously enables efficient and immediate dynamic VN allocation.

Thirdly, this thesis proposed a dynamic VN allocation method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly optimize the network resources even when traffic demands change drastically by applying MADRL for dynamic VN allocation. It can also reduce the agents' constraint violations such as network congestion and server overload and reduce the reallocation such as VM migration by introducing a cooperative element for MADRL. Simulations revealed that the proposed dynamic VN allocation method can reduce the maximum server and link utilization and drastically reduce the constraint violations compared with that of a static VN allocation method under practical-network conditions. In contrast, the evaluation also revealed that the Exhaustive Search (ES) that maximizes the reward at each time does not necessarily maximize the average rewards when the traffic demands fluctuate. Moreover, the computation time of the

proposed method was less than one second, which is significantly shorter than that of ES. As a result, this work revealed that the proposed method simultaneously enables efficient and immediate dynamic VN allocation. Finally, this work evaluated the generalization performance for various traffic demands. The results revealed that the agent training with mixed various traffic models could achieve a high generalization performance for all traffic models.

Fourthly, this thesis proposed a cooperative task-offloading method for multi-cloud and multi-edge networks considering network topology and bandwidth constraints. This method is based on Coop-MADRL. This method can quickly achieve efficient task offloading by learning the relationship between network-demand patterns and optimal task offloading by using deep reinforcement learning (DRL) in advance. This method also introduces a cooperative multi-agent technique, improving the efficiency of task offloading. Evaluations revealed that the proposed method can minimize network utilization and task latency while minimizing constraint violations in less than 1 ms in various network topologies. They also revealed that cooperative learning improves the efficiency of task offloading. This work demonstrated that the proposed method can have generalization performance for various task types by pretraining with many resource-consuming tasks.

The four proposed methods of network resource management based on RL enable rapid calculations of near-optimal resource allocation by learning efficient resource allocation strategies in advance. The near-optimal resource allocation achieved through these methods can accommodate more VNs and tasks using existing network resources. This efficient network management minimizes capital expenditure (CAPEX) and maximizes the profit of telecommunication service providers (TSPs). Moreover, the quick calculations performed by RL enable the maintenance of a near-optimal allocation, even in dynamic fluctuations of network resource demands. This capability allows for timely responses to unexpected changes in demand and network failures, maximizing service quality.

For future works, there are three directions of study for network resource management using MADRL. One direction is expanding the control domain of the proposed methods to new control domains. While this thesis focused on network-resource-integrated control in NFV, other domains are beyond the

scope of this research, e.g., 5th/6th Generation Mobile Communication Systems (5G/6G). In particular, end-to-end network slicing has been focused on recently, which extends the concept of VN to wireless networks, wired networks, and edge/cloud computing. In addition, it is crucial to consider task allocation incorporating emerging technology, e.g., Multi-Access Edge Computing (MEC). The proposed network-resource-integrated control could be further strengthened by incorporating these new control domains. The second direction is developing models and simulations that accurately replicate real-world networks. RL typically learns through trial and error, but real-world networks often do not allow for trial and error. To overcome this challenge, future works should develop techniques to bridge the gap between network simulations and real-world networks. Advanced simulation platforms, such as the Digital Twin platform, can be leveraged to accurately replicate network behavior and provide more effective training and evaluation of RL. The third direction is improving the interpretability of DRL. While this thesis demonstrated the effectiveness of MADRL-based network resource management methods in achieving effective allocation, DRL often needs more interpretability, making it difficult for network operators to understand the decision-making process behind their actions. To address this challenge, future works should focus on enhancing the interpretability of MADRL. This involves developing techniques to explain the reasoning behind agent decision-making, enabling network operators to gain insights into the reasons for agent control and devise appropriate recovery plans in case of agent errors. By improving interpretability, trust in network resource management using MADRL can be increased.

Bibliography

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 2015.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, 2015.

[3] R. Mijumbi, J. Serrat, J.-l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, 2016.

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[5] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. Wilfong, Y. R. Yang, and C. Guo, "Pace: Policy-aware application cloud embedding," in *Proc. IEEE INFOCOM*, 2013, pp. 638–646.

[6] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[7] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, 2016.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, 2012, pp. 2876–2880.

[11] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "MORSA: A multi-objective resource scheduling algorithm for NFV infrastructure," in *Proc. APNOMS*, 2014, pp. 1–6.

[12] Y. Jin, Y. Wen, and C. Westphal, "Towards joint resource allocation and routing to optimize video distribution over future Internet," in *Proc. IFIP Netw. Conf.*, 2015, pp. 1–9.

[13] M. T. Beck and J. F. Botero, "Coordinated allocation of service function chains," in *Proc. IEEE GLOBECOM*, 2015, pp. 1–6.

[14] H. Li, L. Wang, X. Wen, Z. Lu, and J. Li, "MSV: An algorithm for coordinated resource allocation in network function virtualization," *IEEE Access*, vol. 6, pp. 76 876–76 888, 2018.

[15] S. Dräxler, H. Karl, and Z. Á. Mann, "Jasper: Joint optimization of scaling, placement, and routing of virtual network services," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 946–960, 2018.

[16] J. Li, W. Shi, Q. Ye, W. Zhuang, X. Shen, and X. Li, "Online joint VNF chain composition and embedding for 5G networks," in *Proc. IEEE GLOBECOM*, 2018, pp. 1–6.

[17] K. Tsagkaris, G. Nguengang, A. Galani, I. Grida Ben Yahia, M. Ghader, A. Kaloxylos, M. Gruber, A. Kousaridas, M. Bouet, S. Georgoulas *et al.*,

"A survey of autonomic networking architectures: towards a unified management framework," *Int. J. Netw. Manage.*, vol. 23, no. 6, pp. 402–423, 2013.

[18] K. Tsagkaris, M. Logothetis, V. Foteinos, G. Poulios, M. Michaloliakos, and P. Demestichas, "Customizable autonomic network management: integrating autonomic network management and software-defined networking," *IEEE Veh. Technol. Mag.*, vol. 10, no. 1, pp. 61–68, 2015.

[19] A. Stamou, G. Kakkavas, K. Tsitseklis, V. Karyotis, and S. Papavassiliou, "Autonomic network management and cross-layer optimization in software defined radio environments," *Future Internet*, vol. 11, no. 2, p. 37, 2019.

[20] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 2013.

[21] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2018.

[22] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. IEEE NetSoft*, 2015, pp. 1–9.

[23] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Claeys, J. Famaey, and F. De Turck, "Neural network-based autonomous allocation of resources in virtual networks," in *Proc. IEEE EuCNC*, 2014, pp. 1–6.

[24] R. Mijumbi, J.-L. Gorricho, J. Serrat, M. Shen, K. Xu, and K. Yang, "A neuro-fuzzy approach to self-management of virtual network resources," *Expert systems with applications*, vol. 42, no. 3, pp. 1376–1390, 2015.

[25] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM Workshops*, 2019, pp. 879–885.

[26] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1040–1057, 2020.

[27] A. Suzuki and S. Harada, "Safe multi-agent deep reinforcement learning for dynamic virtual network allocation," in *Proc. IEEE GLOBECOM*, 2020, pp. 1–7.

[28] X. Zheng, Y. Zhang, H. Zhang, and Q. Xue, "An RBF neural network–based dynamic virtual network embedding algorithm," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 23, p. e4516, 2019.

[29] C. K. Dehury and P. K. Sahoo, "DYVINE: Fitness-based dynamic virtual network embedding in cloud computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1029–1045, 2019.

[30] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM HotNets*, 2016, pp. 50–56.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAI*, 2016, pp. 2094–2100.

[32] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, "Cooperative task offloading in three-tier mobile computing networks: An ADMM framework," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2763–2776, 2019.

[33] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Trans. Autom. Sci. Eng.*, 2020.

[34] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 624–634, 2020.

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

[36] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Deep reinforcement learning for collaborative offloading in heterogeneous edge networks," in *Proc. IEEE/ACM CCGrid*. IEEE, 2021, pp. 297–303.

[37] W. Hou, H. Wen, H. Song, W. Lei, and W. Zhang, "Multi-agent deep reinforcement learning for task offloading and resource allocation in cybertwin based networks," *IEEE Internet Things J.*, 2021.

[38] S. Ding and D. Lin, "Multi-agent reinforcement learning for cooperative task offloading in distributed edge cloud computing," *IEICE Trans. Inf. Syst.*, vol. 105, no. 5, pp. 936–945, 2022.

[39] Y. Zhang, B. Di, Z. Zheng, J. Lin, and L. Song, "Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2565–2578, 2020.

[40] G. J. Laurent, L. Matignon, L. Fort-Piat *et al.*, "The world of independent learners is not markovian," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.

[41] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv:1801.08757*, 2018.

[42] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 5392–5402.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[44] Y. Fujita, T. Kataoka, P. Nagarajan, and T. Ishikawa, "ChainerRL: A deep reinforcement learning library," in *Proc. NeurIPS Workshop*, 2019.

[45] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symposium Series*, 2015.

[46] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[47] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *Applied Intelligence*, pp. 1–46, 2022.

[48] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, pp. 1–49, 2022.

[49] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[50] F. A. Oliehoek, C. Amato *et al.*, *A concise introduction to decentralized POMDPs*. Springer, 2016.

[51] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv:1706.02275*, 2017.

[52] M. TAN, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. ICML*, 1993.

[53] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[54] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward." in *Proc. AAMAS*, 2018, pp. 2085–2087.

[55] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. ICML*, 2018, pp. 4295–4304.

[56] A. Suzuki, R. Kawahara, M. Kobayashi, S. Harada, Y. Takahashi, and K. Ishibashi, "Extendable NFV-integrated control method using reinforcement learning," *IEICE Trans. Commun.*, vol. E103.B, no. 8, pp. 826–841, 2020.

[57] A. Suzuki, M. Kobayashi, Y. Takahashi, S. Harada, K. Ishibashi, and R. Kawahara, "Extendable NFV-integrated control method using reinforcement learning," in *Proc. IEEE ICC*, 2018, pp. 1–7.

[58] R. Sun and C. Sessions, "Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors," *IEEE Trans. Syst. Sci. Cybern.*, vol. 30, no. 3, pp. 403–418, 2000.

[59] A. Yousafzai, A. Gani, R. M. Noor, M. Sookhak, H. Talebian, M. Shiraz, and M. K. Khan, "Cloud resource allocation schemes: review, taxonomy, and opportunities," *Knowledge and Information Systems*, vol. 50, no. 2, pp. 347–381, 2017.

[60] J. Son and R. Buyya, "A taxonomy of software-defined networking (SDN)-enabled cloud computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, p. 59, 2018.

[61] F. L. Pires and B. Barán, "A virtual machine placement taxonomy," in *Proc. IEEE/ACM CCGrid*, 2015, pp. 159–168.

[62] H. Saito, H. Honda, and R. Kawahara, "Disaster avoidance control against heavy rainfall," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[63] H. Honda and H. Saito, "Nation-wide disaster avoidance control against heavy rain," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1084–1097, 2019.

[64] R. Summerhill, "The new Internet2 network," in *GLIF Meeting*, 2006.

[65] "GLPK," https://www.gnu.org/software/glpk/.

[66] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016.

[67] T. Jaakkola, M. I. Jordan, and S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," in *Proc. NIPS*, 1994, pp. 703–710.

[68] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[69] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola, *Pyomo–optimization modeling in python*, 2nd ed. Springer Science & Business Media, 2017, vol. 67.

[70] D. E. Bernal, Q. Chen, F. Gong, and I. E. Grossmann, "Mixed-integer nonlinear decomposition toolbox for Pyomo (MindtPy)," in *Computer Aided Chemical Engineering*. Elsevier, 2018, vol. 44, pp. 895–900.

[71] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[72] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[73] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.

[74] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. ICML*, 2016, pp. 1928–1937.

[75] A. Suzuki, R. Kawahara, and S. Harada, "Cooperative multi-agent deep reinforcement learning for dynamic virtual network allocation with traffic fluctuations," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 1982–2000, 2022.

[76] ——, "Cooperative multi-agent deep reinforcement learning for dynamic virtual network allocation," in *Proc. ICCCN*, 2021, pp. 1–11.

[77] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1206–1243, 2018.

[78] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *Proc. NSDI*, 2020, pp. 419–434.

[79] E. Oki, *Linear Programming and Algorithms for Communication Networks*. CRC Press, 2012.

[80] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. NIPS Workshop*, 2014.

[81] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 8026–8037.

[82] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft Multi-Agent Challenge," *CoRR*, vol. abs/1902.04043, 2019.

[83] F. Metzger, T. Hoßfeld, A. Bauer, S. Kounev, and P. E. Heegaard, "Modeling of aggregated IoT traffic and its application to an IoT cloud," *Proc. IEEE*, vol. 107, no. 4, pp. 679–694, 2019.

[84] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—survivable network design library," *Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.

[85] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proc. ICML*, 2018, pp. 1407–1416.

[86] Y. Takahashi, K. Ishibashi, M. Tsujino, N. Kamiyama, K. Shiomoto, T. Otoshi, Y. Ohsita, and M. Murata, "Separating predictable and unpredictable flows via dynamic flow mining for effective traffic engineering," *IEICE Trans. Commun.*, vol. 101, no. 2, pp. 538–547, 2018.

[87] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking*, 2015, pp. 1–9.

[88] A. Suzuki, M. Kobayashi, and E. Oki, "Multi-agent deep reinforcement learning for cooperative computing offloading and route optimization in multi cloud-edge networks," *IEEE Trans. Netw. Service Manag.*, 2023.

[89] A. Suzuki and M. Kobayashi, "Multi-agent deep reinforcement learning for cooperative offloading in cloud-edge computing," in *Proc. IEEE ICC*, 2022, pp. 1–7.

[90] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[91] J. Hu, S. Jiang, S. A. Harding, H. Wu, and S.-w. Liao, "Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning," *arXiv:2102.03479*, 2021.

[92] A. Vasiliadis, "An introduction to mean field games using probabilistic methods," *arXiv:1907.01411*, 2019.

[93] D. Chen, Q. Qi, Z. Zhuang, J. Wang, J. Liao, and Z. Han, "Mean field deep reinforcement learning for fair and efficient UAV control," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 813–828, 2020.

[94] L. Li, Q. Cheng, K. Xue, C. Yang, and Z. Han, "Downlink transmit power control in ultra-dense UAV network based on mean field game and deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 15 594–15 605, 2020.

[95] D. Shi, H. Gao, L. Wang, M. Pan, Z. Han, and H. V. Poor, "Mean field game guided deep reinforcement learning for task placement in cooperative multiaccess edge computing," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9330–9340, 2020.

[96] R. A. Banez, H. Tembine, L. Li, C. Yang, L. Song, Z. Han, and H. V. Poor, "Mean-field-type game-based computation offloading in multi-access edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 12, pp. 8366–8381, 2020.

[97] R. Zheng, H. Wang, M. De Mari, M. Cui, X. Chu, and T. Q. Quek, "Dynamic computation offloading in ultra-dense networks based on mean field games," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6551–6565, 2021.

[98] Z. Jian, W. Muqing, and Z. Min, "Joint computation offloading and resource allocation in C-RAN with MEC based on spectrum efficiency," *IEEE Access*, vol. 7, pp. 79 056–79 068, 2019.

[99] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, and P. Zhang, "Energy-aware mobile edge computation offloading for IoT over heterogenous networks," *IEEE Access*, vol. 7, pp. 13 092–13 105, 2019.

[100] L. Tang and H. Hu, "Computation offloading and resource allocation for the internet of things in energy-constrained MEC-enabled HetNets," *IEEE Access*, vol. 8, pp. 47 509–47 521, 2020.

[101] B. Liu, J. Song, J. Wang, H. Sun, and Q. Wang, "Robust secure wireless powered MISO cognitive mobile edge computing," *IEEE Access*, vol. 8, pp. 62 356–62 366, 2020.

[102] L. Wang, H. Shao, J. Li, X. Wen, and Z. Lu, "Optimal multi-user computation offloading strategy for wireless powered sensor networks," *IEEE Access*, vol. 8, pp. 35 150–35 160, 2020.

[103] K. Wang, Y. Zhou, Q. Wu, W. Chen, and Y. Yang, "Task offloading in hybrid intelligent reflecting surface and massive MIMO relay networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 6, pp. 3648–3663, 2021.

[104] J. Xu and J. Yao, "Exploiting physical-layer security for multiuser multicarrier computation offloading," *IEEE Wireless Commun. Lett.*, vol. 8, no. 1, pp. 9–12, 2018.

[105] X. Lai, L. Fan, X. Lei, Y. Deng, G. K. Karagiannidis, and A. Nallanathan, "Secure mobile edge computing networks in the presence of multiple eavesdroppers," *IEEE Trans. Commun.*, vol. 70, no. 1, pp. 500–513, 2021.

[106] Y. Guo, R. Zhao, S. Lai, L. Fan, X. Lei, and G. K. Karagiannidis, "Distributed machine learning for multiuser mobile edge computing systems," *IEEE J. Sel. Topics Signal Process.*, vol. 16, no. 3, pp. 460–473, 2022.

[107] I. Kovacevic, A. S. Shafigh, S. Glisic, B. Lorenzo, and E. Hossain, "Multi-domain network slicing with latency equalization," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2182–2196, 2020.

[108] H. Bai, Y. Zhang, Z. Zhang, and S. Yuan, "Latency equalization policy of end-to-end network slicing based on reinforcement learning," *IEEE Trans. Netw. Service Manag.*, 2022.

[109] Y. Chiang, C.-H. Hsu, G.-H. Chen, and H.-Y. Wei, "Deep Q-learning based dynamic network slicing and task offloading in edge network," *IEEE Trans. Netw. Service Manag.*, 2022.

Publication List

Journal Papers

1. **A. Suzuki**, M. Kobayashi, and E. Oki, "Multi-Agent Deep Reinforcement Learning for Cooperative Computing Offloading and Route Optimization in Multi Cloud-Edge Networks," *IEEE Transactions on Network and Service Management*, 2023. (Early Access)

2. **A. Suzuki**, R. Kawahara, and S. Harada, "Cooperative Multi-Agent Deep Reinforcement Learning for Dynamic Virtual Network Allocation with Traffic Fluctuations," *IEEE Transactions on Network and Service Management*, Vol. 19, No. 3, pp. 1982–2000, 2022.

3. **A. Suzuki**, R. Kawahara, M. Kobayashi, Y. Takahashi, S. Harada, and K. Ishibashi, "Extendable NFV-Integrated Control Method Using Reinforcement Learning," *IEICE Transactions on Communications*, Vol. E103.B, No. 8, pp. 826–841, 2020.

4. **A. Suzuki**, T. Kamioka, Y. Kamakura, and T. Watanabe, "Particle-based Semiconductor Device Simulation Accelerated by GPU computing," *Japan Society for Simulation Technology*, Vol. 2, No. 1, pp. 211–224, 2015.

International Conference Papers

1. M. Iwamoto, **A. Suzuki** and M. Kobayashi, "Deep Reinforcement Learning based Antenna Selection for Cell Outage Compensation," in *Proceed-*

ings of IEEE International Conference on Communications (ICC), May. 2023.

2. M. Iwamoto, **A. Suzuki** and M. Kobayashi, "Optimal VNF Scheduling for Minimizing Duration of QoS Degradation," in *Proceedings of IEEE Consumer Communications & Networking Conference (CCNC)*, Jan. 2023.

3. **A. Suzuki** and M. Kobayashi, "Multi-Agent Deep Reinforcement Learning for Cooperative Offloading in Cloud-Edge Computing," in *Proceedings of IEEE International Conference on Communications (ICC)*, May. 2022.

4. **A. Suzuki**, R. Kawahara and S. Harada, "Cooperative Multi-Agent Deep Reinforcement Learning for Dynamic Virtual Network Allocation," in *Proceedings of 2021 International Conference on Computer Communications and Networks (ICCCN)*, Jul. 2021.

5. **A. Suzuki** and S. Harada, "Safe Multi-Agent Deep Reinforcement Learning for Dynamic Virtual Network Allocation," in *Proceedings of IEEE Global Communications Conference (Globecom)*, Dec. 2020.

6. **A. Suzuki**, M. Kobayashi, Y. Takahashi, S. Harada, K. Ishibashi, and R. Kawahara, "Extendable NFV-Integrated Control Method Using Reinforcement Learning," in *Proceedings of IEEE International Conference on Communications (ICC)*, May. 2018.

7. **A. Suzuki**, T. Kamioka, Y. Kamakura, K. Ohmori, K. Yamada, and T. Watanabe, "Source-induced RDF Overwhelms RTN in Nanowire Transistor: Statistical Analysis with Full Device EMC/MD Simulation," in *Proceedings of IEEE International Electron Devices Meeting (IEDM)*, Dec. 2014.

8. **A. Suzuki**, T. Kamioka, Y. Kamakura, and T. Watanabe, "Full-Scale Whole Device EMC/MD Simulation of Si Nanowire Transistor Including Source and Drain Regions by Utilizing Graphic Processing Units," in *Proceedings of 2014 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, Sep. 11, 2014.

9. **A. Suzuki**, T. Kamioka, H. Imai, Y. Kamakura, and T. Watanabe, "Accelerated parallel computing of carrier transport simulation utilizing graphic processing units," in *Proceedings of 16th International Workshop on Computational Electronics (IWCE)*, Jun. 2013.

Awards

1. Research Award on Information Networks of IEICE, Mar. 2023. (Japanese)
2. Young Researcher Encouragement Award on Information Networks of IEICE, Mar. 2020. (Japanese)
3. Academic Encouragement Award of IEICE, Mar. 2019. (Japanese)
4. Azusa Ono Memorial Award, Mar. 2015.
5. IEEE EDS Japan Chapter Student Award, Feb. 2015.