DYNAMIC RESOURCE COORDINATION TOWARDS RELIABLE AND FLEXIBLE NETWORK SLICING

by

Genya Ishigaki

APPROVED BY SUPERVISORY COMMITTEE:

Jason Jue, Chair

Ravi Prakash

Ramaswamy Chandrasekaran

Jorge A. Cobb

For my mentor, Founder Dr. Daisaku Ikeda, who wrote
"Only labor and devotion to one's mission give life its worth."

For my brother, who taught me the purpose of one's life.

DYNAMIC RESOURCE COORDINATION TOWARDS RELIABLE AND FLEXIBLE NETWORK SLICING

by

GENYA ISHIGAKI, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

August 2021

ACKNOWLEDGMENTS

First and foremost, I express my appreciation to my wife. Your never-ceasing brightness made me realize we can find hope even in the most challenging times of our life. You also taught me the joy of stepping out of my comfort zone through your boldness. It is my fortune to have a partner like you in my life and share all joys and struggles with you.

Needless to say, none of my academic achievements were possible without the inconspicuous support from my parents. I cordially thank you for the absolute trust that you have given to me. Your confidence in me became the source of courage to pursue my degree to this day.

I also want to extend my appreciation to my parents-in-law, who warmly welcomed me to the family. My PhD life became more joyful thanks to your kind support.

I have been very lucky to be educated by many great professors and teachers. I would like to show my deepest gratitude to my advisor, Dr. Jason Jue, who allowed me to come to UTD and trained me throughout the program. Without your compassionate and unchanging support, I would not have been able to enjoy the process towards my degree in this way. I believe I could learn what it means to be professional from you. I will try my best to repay the debts of gratitude I owe you for my entire life.

I wish to express my sincere gratitude to my dissertation committee, Dr. Ravi Prakash, Dr. Ramaswamy Chandrasekaran, and Dr. Jorge Cobb, for their guidance towards my dissertation. When I recall the PhD life at UTD in the future, I will definitely remember all the conversations we had at your labs as one of the most memorable parts of it. I also must thank Dr. Gopal Gupta, who gave me a lot of educational opportunities and encouragement for my future career.

While cherry blossoms bloom in spring, the bloom is only possible with the buds that are prepared in the previous summer and persevered with the severe winter. I would like to thank all the professors at Soka University and teachers at Kansai Soka High School, who

trusted my potential more than I did. In particular, I must thank Dr. Norihiko Shinomiya and Dr. Teruaki Kitano for their ceaseless support over the past ten years.

One of the best things in my PhD life was the precious encounters with amazing friends. I am thankful to all members of the Advanced Networks Research Lab for making my studies at the lab more meaningful and fun. My heartfelt gratitude goes to Riti Gour, Siddartha Devic, Dr. Ashkan Yousefpour, Dr. Jian Kong, Dr. Nannan Wang, and Zhouxiang Wu. Especially, my dissertation was distilled from all the equations that appeared and were wiped out on the whiteboard shared with Riti and Sid. I may forget the proofs of NP-completeness but will remember the time we spent together in front of them.

I also want to express my appreciation for Masaaki Miyashita. You have been not only a good friend but also an inspiration to me since the day we first encountered and learned the Java graph library together in the lab.

I would like to thank Koichi and Akira Tanabe for helping me to settle down in Dallas. I was not able to survive the first year of my PhD without your warm support and encouragement.

I also must thank Dr. Tadamoto Isogai, Josef Gaudiesus, and Erbold Uran for their friendship. The struggles shared with you all became a treasure and foundation of my life.

April 2021

DYNAMIC RESOURCE COORDINATION TOWARDS RELIABLE AND FLEXIBLE NETWORK SLICING

Genya Ishigaki, PhD
The University of Texas at Dallas, 2021

Supervising Professor: Jason Jue, Chair

The increasing demand for a diverse array of network applications entails a more flexible and reliable networking paradigm. *Network slicing* is envisioned to package a set of networking, computing, and storage resources in a coordinated manner, so that the network slice with the tailored set of resources satisfies service requirements unique to each network application. A key enabling technology of network slicing is network softwarization, exemplified by Software Defined Networking (SDN) and Network Function Virtualization (NFV), which enable swift migration of both networking and computing resources. While network slicing sets a conceptual foundation for next-generation networking for diverse applications, the resource coordination mechanism that dynamically operates and manages the resources remains a challenging issue. This is more so when considering the computational complexity induced by the dependency relationship among the softwarized resources and the uncertainty of future network states, such as network failure scenarios and traffic patterns.

This dissertation features four resource allocation problems that collectively facilitate the agile operation and management of end-to-end network slices. The first two problems are related to the reliability aspect of network slicing. In particular, we discuss protection and recovery problems of interdependent network components from a resource allocation standpoint. The third problem deals with a dynamic bandwidth allocation in optical access

networks. The dynamic adjustment of allocated bandwidth assists more effective resource utilization and the accommodation of different types of network services. Furthermore, SDN controller placement, which determines responsiveness to a request for end-to-end resource coordination, is examined as the fourth problem.

The theoretical analysis and proposed algorithms for the problems not only solve the specific resource allocation tasks, but also provide fundamental insights to tackle similar allocation problems under entangled dependency and future uncertainty. In particular, the proposed learning-based approaches project an automated resource coordination system for more effective utilization of network resources in 5G and beyond networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Next-Generation Networking: 5G and Beyond

Network technology has been evolving to accommodate the increasing volume of Internet traffic since its birth in the 1980s. However, the recent emergence of diverse vertical network applications, such as the Internet of Things (IoT) and autonomous driving vehicles, has posed a set of unprecedented challenges to network design, operation, and management. Those challenges and expectations for next-generation networking technologies compel an ideological shift in many aspects of network design and operational methodology due to the uniqueness of the problems. The uniqueness arises from the fact that the next-generation networks are envisioned to support not only the increasing volume of traffic, which had been a central evolving factor till the current-generation networks, but also differentiated classes of network service requirements. This infers the departure from the long-standing philosophy of *one-size-fits-all* network design for a *custom-made* networking approach [50].

One of the most pressing challenges is the coordination of networking and computing resources to provide such tailored networks for each network user [65]. While traditional networks depend on monolithic network designs and proprietary hardware appliances, next-generation networks should be more flexible, dynamic, and, simultaneously, reliable. The development of network softwarization techniques, exemplified by Software Defined Networking (SDN) and Network Function Virtualization (NFV), in the past ten years enables network operators to serve the flexibility and dynamics through the use of commodity networking and computing resources.

5G networks are a precursory realization of such next-generation networks, which own the capability to support a diverse array of network applications [2]. The 5G standard mainly targets three types of network applications that each demonstrate an extreme use-case; namely, Enhanced Mobile Broadband Connectivity (eMBB), Massive Machine Type

Communications (mMTC), and Ultra-Reliable Critical Communication Services (URCC) [81]. eMBB is an extension of the current mobile broadband by extending network coverages, including three-dimensional connectivity. Industrial and IoT applications are supported by mMTC that accept highly dense connectivity. URCC is envisioned to serve for real-time applications such as autonomous driving vehicles and remote medical usages.

The capability of 5G networks hosting different types of network services is enabled by a concept called *network slicing* [20]. Network slicing prepares a virtual network, which consists of virtualized networking, computing, and storage functions, for each network user, as if the user is provided a dedicated physical network tailored for the specific user. Since each network slice is composed to meet the requirements of the network services running on it, network users can experience the guarantee of end-to-end service performance. Figure 1.1 illustrates a simplified 5G network architecture including a network slice spanning from an access network to the core network.

While network slicing sets a successful guideline for the provisioning of custom-made virtual networks, this concept can be implemented only with effective resource coordination that helps network slice providers to profit. Nevertheless, the coordination tends to be complex due to the entangled interactions among virtualized and physical infrastructure resources. Furthermore, more autonomous coordination will become demanding certainly in 6G and beyond networking, as the composition of such network slices become more complex with the dynamics of network traffic and more diverse applications [59, 38]. Therefore, this dissertation tackles multiple resource optimization problems that collectively perform the resource coordination for network slicing. The rest of this chapter describes preliminary information about network slicing and highlights the problems discussed in this dissertation.

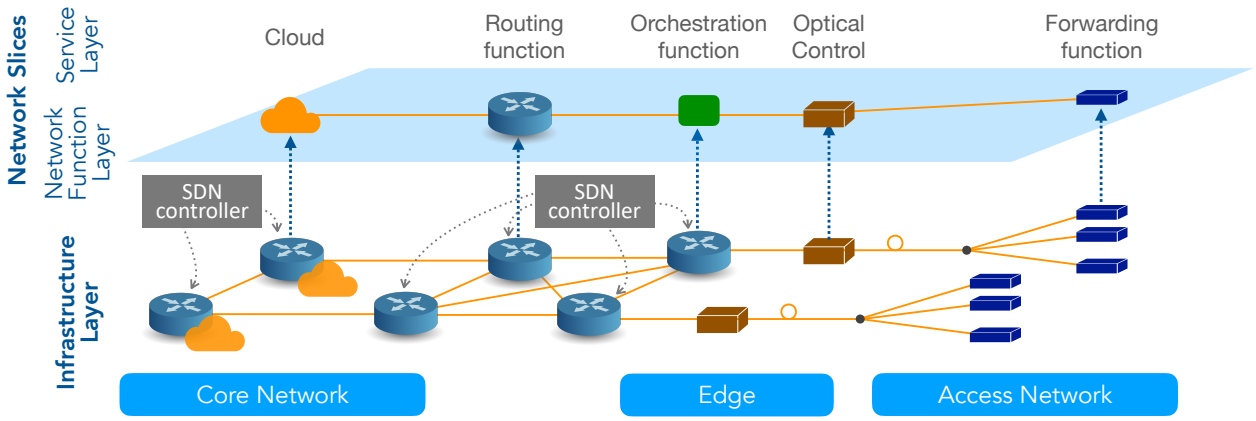

Figure 1.1. 5G Network Architecture: Network slicing over an infrastructure network.

1.2 Key Enablers of Next-Generation Networking

1.2.1 Software Defined Networking (SDN)

Software Defined Networking (SDN) is a networking technology that realizes the programmability of network components through decoupling of the control plane from the data plane of networks [49]. A logically centralized SDN controller monitors and manages the topology and routing of a data plane, while the routers and switches in the data plane naively forward datagrams based on the policy delivered from the SDN controller.

The centralization and decoupling provide a network with more manageability and flexibility. In traditional networks, the deployment of a change in routing policy required significant effort, since the routers and switches update their routing tables in a distributed manner. In contrast, an update on routing policy can be immediately communicated from a controller to all routers and switches with SDN, which facilitates more dynamic configurations of networks. This SDN architecture also supports faster failure recovery as an SDN controller can determine an optimal recovery plan based on a global view of a network after identifying a link or node failure.

OpenFlow is one of the most popular standardized SDN protocols that define a communication protocol between an SDN controller and SDN switches [36]. This southbound interface standardization absorbs the differences among heterogeneous routers and switches from different vendors. SDN also provides a northbound API for the controllers, so network operators can implement new routing and failover logic. This programmability promotes the use of an advanced networking operation with a complex algorithm, as a network operator does not need to translate their complex logic to the actual router configurations.

1.2.2 Network Function Virtualization (NFV)

Network functions can be seen as software appliances when employing Network Function Virtualization (NFV) [1]. Traditionally, network functions were implemented on dedicated hardware in an entangled way. NFV makes a clear separation between the logical functionality and hardware requirements so that Virtualized Network Functions (VNFs) can run on general-purpose servers.

NFV realizes another level of flexibility in resource allocation, which eventually enhances the scalability of a network. As a network function is a software module with NFV, it becomes easier to move the location of function deployment across a network. It is beneficial to meet a specific service requirement in latency. Moreover, the volume of a network function can be easily scaled up or down depending on the demand for the function, which further accelerates the dynamic reconfigurations of a network. The dynamic reconfigurations are only made possible through NFV, since the scaling can be conducted by software deployment on general-purpose servers, instead of hardware deployment.

European Telecommunication Standards Institute (ETSI) defined NFV management and orchestrator (NFV-MANO) to provide an architectural view of NFV [16, 80]. Multiple subfunctions of NFV-MANO collaboratively manage the life-cycle of VNFs. In particular, Virtualized Infrastructure Managers (VIMs) manages the abstraction of general-purpose hardware

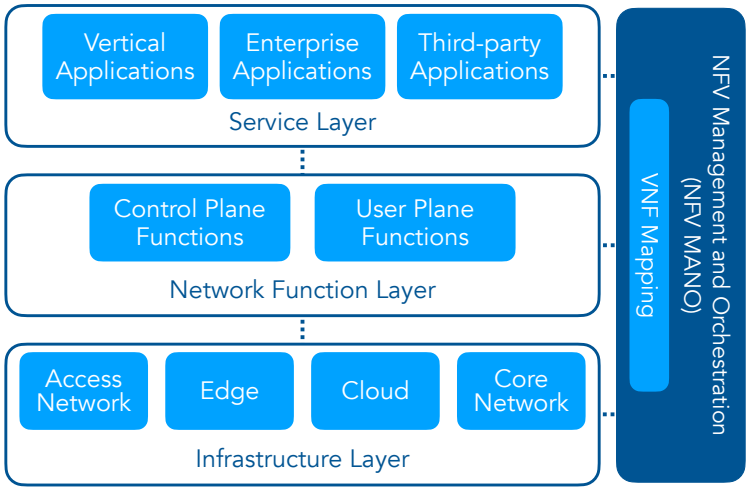

Figure 1.2. Network Softwarization: A NFV MANO coordinates the mapping between network infrastructure and network functions to provide network services [20].

to host VNFs, which provides Network Function Virtualization Infrastructure (NFVI), and VNF Managers (VNFMs) are in charge of software (VNF) deployment. Figure 1.2 illustrates the logical layering used by NFV-MANO.

1.2.3 Network Slicing

Network slicing is an advanced network virtualization concept where virtualized computation, communication, and storage resources are packaged as a network slice tailored for a specific network service [20, 73, 81]. The dedication of resources in a network slice enables one physical network to accommodate multiple network tenants who each provide a network service with different performance requirements.

While network slicing is an extension of Network Virtualization (NV) and NFV, it is supposed to provide additional features that have not been specified in NV or NFV to accommodate diverse network applications in the 5G and beyond era. One of the additional features is *slice isolation* which ensures performance guarantees for each network slice independent from states of the other network slices in the same physical network [73]. In other words, failure, security incidents, or load increase in coexisting network slices must

not affect the performance of the other slice. Another important feature is *abstraction of service requirements* from each tenant [20]. This aspect is closely related to Infrastructure-as-a-Service (IaaS), which provides high-level APIs to configure computing, networking, and storage infrastructures. Usually, slice tenants are the entities who want to run a specific network service to their customers and are not necessarily trained for the detailed configuration of networking infrastructure. With this assumption, a network slice provider should provide high-level APIs, so that such tenants can describe their requirements in a form of Service Level Agreements (SLA) without specialized knowledge. Therefore, it is the responsibility of a slice provider to translate a given SLA into an actual resource configuration. Furthermore, service requirements for a network slice tend to be specified as an *end-to-end* SLA, since slice tenants intend to guarantee certain service performance at the application level.

Those features desired for network slicing enable slice tenants to provide more reliable and scalable network services. The isolation of network slices naturally brings robustness against network attacks and failures. Also, slice tenants do not need to worry about the details of a mechanism to ensure a certain SLA, as they can use high-level APIs to specify the SLA. Furthermore, the high-level view of a provided network slice provides more flexibility in resource scaling. As many network services experience temporal and geographical demand dynamics, it is expected for network slicing to serve easily reconfigurable network infrastructures. From the perspective of network slice providers, they could increase the revenue when utilizing the physical resources in an efficient way. In particular, the deployment of general-purpose physical infrastructures with virtualization techniques opens a way to adjust their virtual resources for popular services and high-profit types of tenants.

1.3 Unique Challenges towards Reliable and Flexible Network Slicing

1.3.1 Interdependency

As illustrated in Figure 1.2, a benefit of the virtualization techniques is the decoupling of network functions and infrastructure hardware. Due to the clear separation between the network function layer and infrastructure layer, a network operator can adjust network resources to satisfy varying demands from different users.

Nevertheless, the decoupling also brings additional complexity into networks by introducing dependency among network components. The dependency of the network function layer on the infrastructure layer is one the most obvious relation that introduces a new aspect of network fragility. It is possible for multiple network slices to simultaneously experience network failures due to a failure of a single infrastructure node, when the infrastructure node hosts multiple network functions deployed on the network slices. This type of failure propagation has been studied as a Shared Risk Resource Group (SRRG) [40]. Furthermore, the opposite direction of dependency where the infrastructure layer requires a connection to the network function layer is also observed, especially when the control plane functions are deployed. This is because the control or orchestration functions in the network function layer are necessary for infrastructure resources to be coordinated to fulfill the designated functionality [39].

As resource virtualization is a core technology to realize reliable and flexible network slicing, it is necessary to consider the interdependency among the network function and infrastructure layers for slice resource coordination. An *interdependent network* is an effective representation in which the dependency relationship among the virtualized and physical resources is abstracted as directed edges [52]. The graph-theoretic formalism provides a framework to discuss the fragility caused by the interdependency.

1.3.2 Uncertainty of Future Demand and Available Resources

One of the biggest expectations for next-generation networking is the elasticity of resource provisioning. Network slice tenants expect to save the cost related to extra resources reserved for unknown future demand, assuming the ability of network slices to scale on demand. From the perspective of a slice provider, more sophisticated coordination is required to guarantee such scaling and the isolation of network slices on the same physical network.

The uncertainty of future traffic patterns has been a challenging issue in the networking field. While many traffic estimation methods are available for long-term statistical patterns, it is still difficult to estimate short-term traffic trends. The diversity in network applications in 5G and beyond networks will add additional complexity to the estimation problem, because the behaviors of network users become more complicated due to their mobility, types of communications, and dynamic resource reservation.

In addition to the uncertainty of future demand, uncertainty about the availability of resources could be another challenge. When network slice providers are only given incomplete information about the amount and timing of future resource dispatches, they cannot schedule their resource allocation actions in advance. In this type of scenario, they need to either estimate the resource availability to solve the resource scheduling problem or come up with a robust allocation solution that accommodates multiple scenarios within a certain range of resource availability.

1.4 Research Statement

A fundamental question underlying this dissertation is about the tradeoff in resource allocation: How does a network operator balance the maximization of resource utilization and the reservation of spare capacity for future demands? Even though the tradeoff has been a paramount agenda of resource optimization research per se, the new features introduced for Next-Generation Networking present additional challenges to answer the question.

Table 1.1. Problem Space Discussed in Each Research Problem.

Problem	Problem Space	Chapter
Problem I	Interdependency (Protection)	Chapter 2
Problem II	Interdependency (Recovery), Resource Uncertainty	Chapter 3
Problem III	Traffic Uncertainty	Chapter 4
Problem IV	Coordination towards Flexible Network Slices	Chapter 5

In particular, this dissertation focuses on resource optimization under the complex dependency of network components and uncertainty over future network and resource states, which are commonly observed in recent networking paradigms. While discussing different types of resource allocation problems appearing in 5G and beyond networks, we propose a set of resource allocation algorithms that collectively realize the reliable and flexible network slicing for diverse network services.

Each resource allocation problem discussed in this dissertation is summarized below. Chapter 2 and 3 deal with resource allocation for the protection and recovery of networks with an emphasis on the interdependency among infrastructure resources and virtualized network components. A bandwidth allocation algorithm is proposed for optical access networks in Chapter 4, considering the uncertainty of future traffic. In Chapter 5, we discuss a placement problem of SDN controllers that each manage a domain of physical networks and coordinate resource allocation for end-to-end network slices.

1.4.1 Problem I: Reliable Network Design considering Interdependency

The effective use of virtualization techniques requires sophisticated management of relationships among softwarized modules, since a set of the modules and hardware hosting them collaboratively provide network service functionality. Several prior research works demonstrate that poor management of such inter-module relations could cause unique fragility of networks called *cascading failure* [52, 39]. In a cascading failure, a failure of a part of a

network propagates to the rest of the network along with the dependency relations among the modules and hardware. The focus of my work is to (i) identify dependency structures that can mitigate the impact of cascading failures and (ii) propose a protection method that constructs such robust dependency structures by appropriate resource allocation. In Chapter 2, a combinatorial algorithm to improve the robustness of interdependent network components is presented based on the analytical work on such robust dependency structures.

1.4.2 Problem II: Recovery Resource Allocation for Interdependent VNFs

In addition to network protection, the recovery of failed dependent network components is a challenging task that involves additional complexity compared to the recovery of independent network components. In particular, the challenge becomes evident when the recovery is conducted progressively due to limited repair resources. The ordering of resource allocation in the recovery phase influences the interim computation and communication capability. This is because certain network components require not only repair resources but also the working operation of other network components in order to function properly. This combinatorial utility on repair resource allocation makes the recovery problem more complex than typical network recovery problems. Hence, our work deals with a progressive recovery problem under limited resources in networks with interdependent VNFs. In Chapter 3, a Deep Reinforcement Learning (Deep RL)-based solution for the progressive recovery problem is described along with the intractability analysis of the problem. Furthermore, a set of our theorems show that a broader class of progressive recovery problems can be reduced to the cases solved in this work.

1.4.3 Problem III: Dynamic Bandwidth Allocation for Optical Access Networks

While network resources can be statically allocated to each network slice, it is further beneficial for both slice tenants and slice operators to dynamically adjust the allocation and utilize

the spare capacity efficiently. This type of network slice is called an *elastic* network slice. An indisputably important problem is the dynamic resource allocation under uncertain traffic trends. Furthermore, the profitability and scalability will be dominated by such allocation strategies when network functions on the slices collaboratively realize service functionality. This work aims to propose a resource allocation methodology to deal with the tradeoff between over- and under-provisioning for network slices. Chapter 4 discusses a dynamic bandwidth allocation problem in optical access networks, which is the initial step towards the elasticity of end-to-end network slices from the network edge to the core network. The dynamic allocation is realized by a learning algorithm called Online Convex Optimization (OCO) that guarantees performance bounds in terms of the hindsight optimum called *regret*.

1.4.4 Problem IV: Controller Placement towards Scalable SDN

The end-to-end service requirements for network slices demand resource coordination across multiple network domains from access networks to the core network. Assuming the deployment of SDN technologies at each domain, the coordination is conducted through a logically centralized orchestration function that communicates with all the SDN controllers responsible for each domain. In practice, the logical orchestrator function is implemented through exchanges of orchestration messages among the SDN controllers. Chapter 5 discusses an SDN controller placement problem that minimizes the communication latency of the exchanges of orchestration messages among multiple controllers. Inspired by an approximation algorithm for the knapsack problem, we propose a greedy controller placement algorithm that reduces the distance among the controllers.

CHAPTER 2

IMPROVING THE SURVIVABILITY OF CLUSTERED INTERDEPENDENT NETWORKS BY RESTRUCTURING DEPENDENCIES

The interdependency between different network layers is commonly observed in communication networks adopting the dissociation of logic and hardware implementation, such as Software Defined Networking and Network Function Virtualization. This chapter[1] discusses a network design problem to improve the survivability of existing interdependent networks by restructuring the dependency relationships between the network function and infrastructure layers. A characteristic of the proposed algorithm is that the continuous availability of the entire system is guaranteed during the restructuring of dependencies by the preservation of certain structures in the original networks. Our simulation results demonstrate that the proposed restructuring algorithm can substantially enhance the survivability of interdependent networks, and provide insights into the ideal allocation of dependencies.

2.1 Introduction

Many network systems encompass layering and integration of the layers in both explicit and implicit manners. For example, Software Defined Networking (SDN) decouples the control logic from forwarding functions to realize the flexibility and agility of communication networks. Also, Network Function Virtualization (NFV) involves the separation of network

[1]The content of this chapter is based on the following earlier works.

function logic from hardware. The concept of separating logic from hardware implementations is also commonly adopted in Cyber Physical Systems (CPS), such as smart grids, in which computing capability manages physical entities.

The dissociation of logic and functions, which is effective for system flexibility, has accelerated the amount of layering and obscure dependencies in network systems. The work [57] on software defined optical networks points out the dependency of logical nodes on physical nodes that provide physical paths for connections among logical nodes, as well as the dependency of physical nodes on the logical nodes through SDN control messages, which define the operations of the physical nodes. Similarly, it is revealed that NFV embraces the interdependency between Virtual Network Functions (VNF) and physical servers hosting the VNFs, when a virtualization orchestrator is recognized as one of the VNFs [39]. Furthermore, the integration of a control information network and an electricity network seen in smart grids is a typical example of the interdependency of two different layers in CPSs [51]. This tendency of layering and collaborative functionality of layered networks is likely to be more evident for next-generation network systems.

However, it has been revealed that certain types of dependencies between different layers of networks can deteriorate the robustness of the entire interdependent system [61]. Consecutive multiple failure phenomena called *cascading failures* exemplify the unique fragility of such network systems. In networks without interdependencies, a failure would influence a certain part of a network. Nonetheless, in networks with interdependencies, some nodes that are not directly connected to the failed portion can become nonfunctional due to the loss of service provisioning from nodes in other layers, which are directly influenced by the initial failure.

Figures 2.1 - 2.4 show an example of such a cascading failure, which starts as a single node failure of v_1 and results in the entire network failure. Suppose that a network G_1 consists of physical servers v_1 and v_1', and G_2 represents logical computing nodes v_2 and v_2' hosting

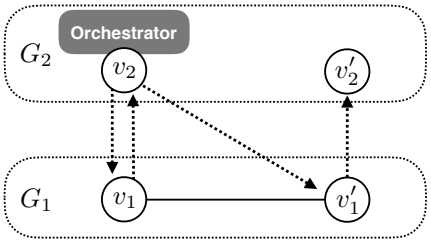

Figure 2.1. An interdependent network with two constituent graphs representing physical and logical network.

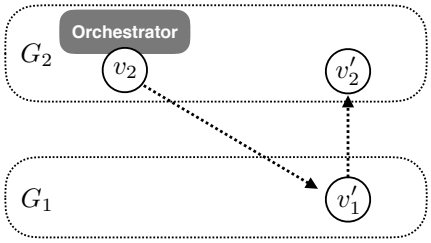

Figure 2.2. Initial failure at a physical server v_1.

Figure 2.3. Cascading failure affecting a logical node v_2.

Figure 2.4. Cascading failure affecting a physical server v_1'. The entire network becomes nonfunctional.

VNFs. The orchestrator, which coordinates the mapping between physical and logical layers, is realized as one of the VNFs on v_2. The arcs from G_1 to G_2 $((v_1, v_2), (v_1', v_2'))$ illustrate the dependency of NFVs or computing nodes on the physical servers, while the arcs from G_2 to

G_1 $((v_2, v_1), (v_2, v_1'))$ indicate the dependency of physical servers on a logical node in terms of the flow of coordination messages from the orchestrator to the physical servers. When the physical server v_1 fails, the logical node hosting the orchestrator v_2 loses its dependent physical node v_1, and becomes nonfunctional. This induces another loss of the dependent node of v_1', and eventually, the single node failure causes a failure of the whole network.

Cascading failures can also lead to the malfunctioning of CPSs. In fact, it has been reported that some major electricity outages in smart grids, such as the 2003 nation-wide blackout in Italy [7], and the 2004 blackout over 8 states in the US and 2 provinces in Canada [4], were due to cascading failures induced from poorly designed dependencies between the electricity network and control information network.

Many contributions have been made since the first theoretical proposal on the cascading failure model by Buldyrev et al. in 2010 [9]. The pioneering works [9, 22] focus on analyzing the behavior of cascading failures rather than proposing design strategies. In contrast, some following works identify vulnerable topologies in interdependent networks to avoid such fragile structures in the design phase by investigating the relationship between node degree and failure impacts [68], or evaluating the importance of nodes exploiting the algebraic expression of dependencies [58]. Furthermore, other works propose design strategies in more realistic models to consider the impact of failures caused by a single component [84], integrated factors within and between layers [56], or the heterogeneity of nodes in each layer [64].

This paper discusses a design problem for interdependent networks to improve their survivability, which is a measure of the robustness against a whole network failure, by modifying an existing network topology. The contribution that contrasts our work with other related works is the consideration of existing network facilities. Our method is aimed at redesigning a relatively small part of the existing network to enhance the survivability so that the entire network remains operational even during the restructuring process. To realize this

continuous availability, a special type of dependency, whose removal does not influence the functionality of the entire system, is identified in the first step of our restructuring method. Our heuristic algorithm increases the survivability of entire systems by the relocations of these dependencies. While our previous work [30] allows a node to have dependencies with any nodes in the other layer, this paper extends the model by considering geographical, economic, or logical accessibility of provisioning by nodes. These constraints are represented as clusters of nodes, and an interdependent network is modeled as a directed graph consisting of multiple clusters. The membership of a node in a specific cluster imposes restrictions on the nodes to which the node can provide support, and the nodes from which the node can receive support. Hence, possible modifications to the dependencies between nodes would vary, depending on the cluster to which a node belongs. Finally, our method is evaluated by simulations in different pseudo interdependent networks.

2.2 Related Works

Most of the preceding works on interdependent networks attempt to analyze the behavior of cascading failures in well-known random graphs, which have certain characteristics in degree distributions and underlying topology [9, 22]. Those works analyze the propagation of failures based on percolation theory developed in the field of random networks. Following the directions shown by a seminal work by Buldyrev et al. in [9], more general models are discussed in [22].

The works [68, 58, 56, 64, 84] focus on the design aspect of interdependent networks. The relation between the impact of failures and interdependencies is empirically demonstrated to decide appropriate dependency allocations in [68]. A method to evaluate the importance of nodes in terms of network robustness is proposed in [58] by introducing a novel representation of interdependencies based on boolean algebra. This evaluation enables network operators to prioritize the protection of the nodes that contribute more to the robustness of the network.

In [56], the authors consider dependency relations not only between layers but also within a single layer. Combining multiple factors that make a node nonfunctional, their method adjusts the dependency of a node on the other nodes. The work in [64] also considers the influence within a single-layer, supposing the heterogeneity of nodes. In this model, a network can have different types of nodes such as generating and relay nodes. Zhao et al. [84] formulate an optimization problem enhancing the system robustness, defining Shared Failure Group (SFG), a group of nodes that can simultaneously fail due to a cascading failure initiated by the same component.

Another branch of interdependent network research is recovery after failures [63, 21, 6, 53, 83, 41]. The works in [63, 21, 6] analyze the behaviors of failure propagations when each node performs local healing, where a functioning node substitutes for the failed node by establishing new connections with its neighbors. The speed of further cascades and resulting network states are revealed by percolation theory [63, 21] or steady state analysis in the belief propagation algorithm [6]. Also, resource allocation problems, which consider the different roles of network nodes are discussed in [53, 83, 41]. The order of assigning repairing resources is a critical problem during the recovery phase when the amount of available resources is limited. The works in [53, 83] propose node evaluation measurements to decide the allocation, while an equivalent problem in the phase diagram is discussed in [41].

Our work proposes a method to improve the survivability of interdependent networks, following the survivability definition in [52]. Our work would be classified into the category of protection design methods before failures. Specifically, the proposed method is exploited in a redesign process of an existing network to enhance the survivability, while the existing works [68, 58, 84, 56, 64] discuss the initial design of an entire network. Our protection method, considering the functionality during the redesign, would reduce the cost of survivability improvement in contrast to the entire reconstruction of the systems.

2.3 Modeling and Motivating Example

In this section, we present a mathematical model for describing interdependent networks, and we present a motivating example of our method. Section 2.3.2 summarizes related work [52] defining the survivability for interdependent networks, which we adopt to evaluate the networks.

2.3.1 Network Model

An interdependent network consists of k constituent graphs $G_i = (V_i, E_{ii})$ $(1 \leq i \leq k)$ and their interdependency relationships, which are defined by sets of (directed) arcs A_{ij} $(1 \leq i, j \leq k,\ i \neq j)$ representing the provisioning between a pair of nodes in different graphs. Edges in $E_{ii} \subseteq V_i \times V_i$ are called *intra*-edges because they connect pairs of nodes in the same network. In contrast, arcs in $A_{ij} \subseteq V_i \times V_j$ $(i \neq j)$ are called *inter-* or *dependency* arcs. If there exists an arc $(v_i, v_j) \in A_{ij}$ $(v_i \in V_i, v_j \in V_j)$, it means that a node v_j has dependency on a node v_i. The node v_i is called the *supporting* node, and v_j is a supported node. A node v is said to be *functional* if and only if it has at least one functional supporting node.

When an interdependent network is logically partitioned, each constituent graph G_i has a clustering function $\kappa_i : V_i \longrightarrow \{1, 2, ..., \gamma_i\}$, where $\gamma_i \in \mathbb{N}$ is the number of clusters in $G_i = (V_i, E_{ii})$. Then, a graph $I_i^x = (W_i^x \subseteq V_i, E_{ii}(W_i^x))$ induced by a node set $W_i^x = \{v \mid \kappa_i(v) = x\ (1 \leq x \leq \gamma_i)\}$ is called a *cluster*. Note that this definition insists that a node is in exactly one cluster.

In order to emphasize the dependency between constituent graphs, an interdependent network can be represented as a single-layer directed graph $G = (V, A)$, where $V := \bigcup_i V_i$, and $A := \bigcup_{\{(i,j)|i \neq j\}} A_{ij}$ by abbreviating intra-edges. With this notation, a node v is said to be *functional* if and only if $\deg_{\text{in}}(v) \geq 1$. Note that all the discussions in the rest of this paper follow this single-layer graph representation.

Additionally, we introduce a different notation of arcs with respect to their source nodes. Let $A(v) \subseteq A$ represent a set of arcs whose source node is $v \in V$. To identify each arc during the restructuring process, where some arc temporarily loses its destination, each arc is denoted as $(v, \cdot)_m$ $(m = 1, ..., \deg_{\text{out}}(v))$. The index m is a given fixed identification number for each arc in $A(v)$. Hence, every arc in A can be specified by providing source node v and its identification number m.

A set of constituent graphs is totally ordered by the number of nodes that are the source of at least one intra-arc: $|V_i^{\text{out}}|$, where $V_i^{\text{out}} := \{v \in V_i \mid A(v) > 0\}$. A constituent graph that has the least number of nodes with outgoing arcs is named the minimum supporting constituent graph G_i: $|V_i^{\text{out}}| \leq \min_j |V_j^{\text{out}}|$.

2.3.2 Survivability of Interdependent Networks

Parandehgheibi et al. [52] propose an index that quantifies the survivability of interdependent networks against cascading failures exploiting the *cycle hitting set*, and they prove that the computation of the survivability is NP-complete. They show that a graph needs to have at least one directed cycle in order to maintain some functional nodes; in other words, the existence of one cycle prevents an interdependent network from its entire failure. Thus, the survivability of interdependent networks is defined as the cardinality of the minimum cycle hitting set whose removal brings non-functionality for the entire network. Note that a cycle hitting set S is a set of nodes such that any cycle $C = (V(C), E(C))$ in a given graph $G = (V, A)$ has at least one node in the hitting set: $S \cap V(C) \neq \varnothing,\ \forall C \in \mathcal{C}(G)$, where $\mathcal{C}(G)$ is the set of all cycles in the given graph. This definition implies that the entire failure of an interdependent network occurs when the corresponding graph becomes acyclic. Let $H(G)$ denote a cycle hitting set with the minimum cardinality: $|H(G)| := \min_{S \in \mathcal{S}} |S|$, where $\mathcal{S}$ is the set of all the cycle hitting sets in G. Formally, the survivability of an interdependent network G is the cardinality of the minimum cycle hitting set, $|H(G)|$.

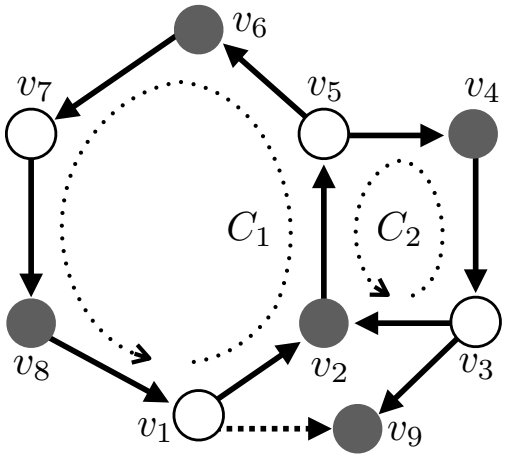

Figure 2.5. Graph G with (v_1, v_9).

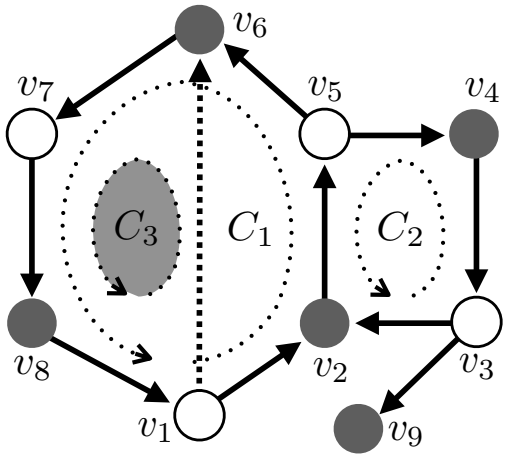

Figure 2.6. Graph G' with (v_1, v_6).

2.3.3 Motivating Example

Adopting the survivability definition shown above, improving survivability would be equivalent to increasing the number of disjoint cycles in a graph. Figures 2.5 and 2.6 show an example comparing two similar interdependent networks.

In graph G in Figure 2.5, there exists two cycles: C_1 and C_2. If v_2, which is in both $V(C_1)$ and $V(C_2)$, becomes nonfunctional because of a failure, all the nodes in G eventually lose their supporting nodes and become nonfunctional: $H(G) = \{v_2\}$. On the other hand, no single node failure can destroy all the three cycles in G' in Figure 2.6, while a two-node failure can make it acyclic (e.g. $H(G') = \{v_2, v_7\}$). Therefore, the graph G' is more survivable than G, since $1 = |H(G)| < |H(G')| = 2$, although they differ only in the destination node of one dependency arc ((v_1, v_9) in G or (v_1, v_6) in G'). Supposing that G is an existing topology of a network, a method that relocates (v_1, v_9) to (v_1, v_6) can achieve an enhancement of the survivability.

2.4 Problem Formulation

2.4.1 Assumptions

This paper deals with the case in which interdependent networks have two types of homogeneous constituent networks with identical dependencies ($k = 2$). However, our discussion with the restriction on k can be easily extended to more general cases. In more advanced network models, each constituent network can have different types of nodes, such as independently functional generating nodes and relay nodes, which need provisioning from a generating node via paths of intra-edges [64]. Nevertheless, for simplicity, this work follows the assumption in [52] that each node in a constituent network is directly connected to a reliable conceptual generating node by a reliable edge (homogeneous constituent graphs). Moreover, it is assumed that each supporting node provides a unit amount of support that is enough for a supported node to be operational (identical dependencies), following the same model in [52].

Additionally, this paper presumes that each cluster x receives some support from at least one of the clusters that are supported by cluster x. In other words, this presumption excludes the case that a cluster does not receive provisions from any of the clusters that the cluster is supporting.

2.4.2 Requirement Specification

One aspect contrasting our scheme to other works is the consideration to improve the survivability of existing interdependent networks by changing some topological structures. Because all the nodes need to remain functional even during the relocations of dependency relations, it is necessary to avoid the loss of all supporting nodes for any node at any stage of the restructuring. In other words, each node needs to be survivable from a cascading failure, which requires the direct or indirect support by the nodes in directed cycles. This constraint is formally represented as the following rule for the live restructuring.

1. Every node remains reachable from a node in a directed cycle via at least one directed path at any stage of the restructuring.

In addition to guaranteeing the continuous availability, the amount of provisioning provided by each supporting node should remain the same after the restructuring in order to consider the capability of each node. The capability could be, for example, the limit on electricity generation, computation performance, or the number of ports available.

2. The number of supports that a node provides must remain less than or equal to its original provisioning capability.

Furthermore, depending on which cluster a node in graph G_i belongs to, the node has a constraint on clusters in G_j that it can support. The constraint is given by a supportability function $\sigma_{ij} : V_i \longrightarrow 2^{\gamma_j}$, where 2^{γ_j} is the power set of the cluster indices in a constituent network G_j. This means that a node $v\,(\in V_i)$ can provide its support to the nodes in the clusters of G_j given by the supportability function. This specification corresponds to the geographical, economic, or logical constraints on the accessibility of supports from a node to specific groups of nodes. For example, it is impossible for information control node v to have electricity supply from node u if v and u are geographically far apart or managed by different administrative institutions. The geographical or administrative domain is shown as a cluster in each constituent graph, and dependency relations of the nodes should be closed within a set of permitted nodes, which are geographically close, or managed by the same company or allied companies, since each cluster should be independent from the outsiders. This constraint relating to network clustering is simply expressed as follows.

3. All the provisionings from a node u are directed towards the nodes in the clusters that u can support, as designated by the supportability function σ_{ij}.

2.4.3 Clustered ΔH Problem

This section formulates the clustered ΔH problem, which is aimed at enhancing the survivability of a given interdependent network with clusters by restructuring dependency relationships, considering the continuous availability, supporting capability, and clustering constraint of each node.

Considering the continuous availability of an existing network during restructuring leads to the formulation of a gradual reconstruction problem, where no relocation of two or more different arcs is conducted at a time. Each phase relocating one arc is named a *step*. Let $G^s = (V, A^s)$ denote the graph representing the interdependent network topology at step s. The improved interdependent network G^{s+1} after step s consists of a node set V, which is the same node set as in graph G^s, and an arc set A^{s+1} amended by the relocation of an arc $(u, v) \in A^s$ to (u, v'), where $v' \in V$ is a new destination for the arc (u, v).

The clustered ΔH problem is to maximize the difference in survivability between a given interdependent network, which is recognized as G^0, and the resulting network after a sequence of consecutive improvements. The resulting network is represented as G^f, where f denotes the step at which the last arc relocation is completed. Formally, the objective is to maximize the difference between $|H(G^0)|$ and $|H(G^f)|$, which is defined as ΔH.

Problem 1 (Clustered ΔH Problem)**.** For a given $G^0 = (V = \bigcup_i V_i, A^0)$, the number of clusters $\gamma_i \in \mathbb{N}$ in each constituent graph G_i, a clustering function $\kappa_i : V_i \longrightarrow \{1, 2, ..., \gamma_i\}$ for each constituent graph G_i, and supportability functions $\sigma_{ij} : V_i \longrightarrow 2^{\gamma_j}$, maximize $\Delta H := |H(G^f)| - |H(G^0)|$, where $G^{s+1} = (V, A^{s+1})$ $(0 \leq s \leq f - 1)$ is obtained by the relocation of the destination of a single arc in A^s: $A^{s+1} = A^s \setminus (u, v) \cup (u, v')$, satisfying

1. $\deg_{\text{in}}(v)_{G^s} \geq 1 \quad \forall v \in V$,

2. $\deg_{\text{out}}(v)_{G^{s+1}} = \deg_{\text{out}}(v)_{G^s} \quad \forall v \in V$,

3. $\kappa_j(v \in V_j) \in \sigma_{ij}(u \in V_i) \ \ \forall (u,v) \in A^s$.

These three conditions correspond to the three rules described in Section 2.4.2. The second and third conditions are easily derived from the corresponding rules. Lemma 1 shows the equivalence of the condition 1 and Rule 1.

Lemma 1. When $\deg_{\text{in}}(v)_G \geq 1$ $(\forall v \in V)$ in a connected directed graph $G = (V, A)$, (a) G has at least one directed cycle, and (b) any node $v \in V$ is reachable from a node $u \in V$ that is contained in a directed cycle.

Proof. $\deg_{\text{in}}(v)_G \geq 1$ $(\forall v \in V)$ insists that any node v has at least one parent v'. The path $v \leftarrow v' \leftarrow ...$ composed by repeating the trace of parents can be acyclic until the length of the path is $|V-1|$. However, the $|V|$th node must have at least one parent from the assumption. Thus, the pigeonhole principle indicates that it is necessary that the path forms a directed cycle. □

2.4.4 Problem Analysis

This section provides the analysis on the trivial optimal case of the clustered ΔH problem with a special setting, where each of constituent graph consists only of one cluster. Let $\rho((u,\cdot)_m)$ denote the number of relocations that arc $(u,\cdot)_m \in A$ experienced during the restructuring process. Note that $\sum_{u \in V} \sum_{m=1}^{\deg_{\text{out}}(u)} \rho((u,\cdot)_m) = f$.

From the definition, the optimum survivability cannot exceed the number of supporting nodes, which each have at least one outgoing arc, in the minimum supporting constituent graph G_i. This is because a set of such nodes covers all the directed cycles in an interdependent network G. This observation implies that the optimum survivability is achieved when every node $v_i \in V_i$ of G_i has an injective mapping to a node in V_j $(j \neq i)$. In other words, for each node v_i in G_i, there exists at least one unique disjoint cycle whose length is 2 with v_j in G_j. The following lemma gives a sufficient condition to reach the ideal state by repeated relocations while preserving the problem constraints.

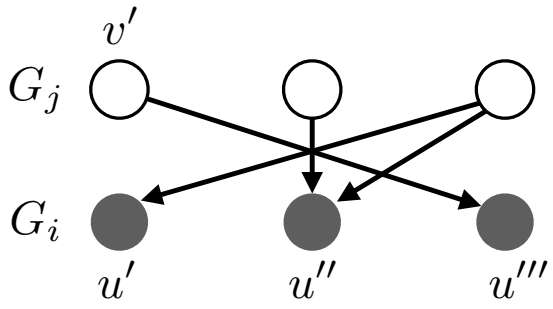

Figure 2.7. Original Dependencies, where (v', u') is missing. Note that this figure only shows A_{ji}. The symmetric discussion can be done for A_{ij}.

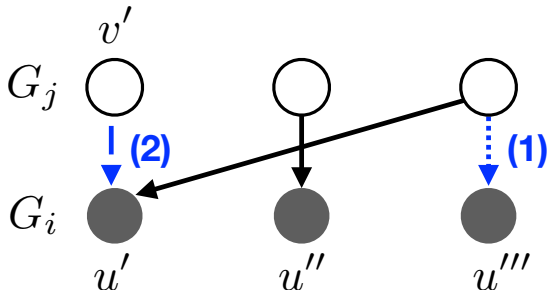

Figure 2.8. Relocation Steps (1) to maintain the functionality of u''', and (2) to form a length-2 cycle with v' and u'.

Lemma 2. When the number of relocations for each arc $\rho((u, \cdot)_m)$ is not upper bounded, in order to have the optimum restructuring, it is sufficient that the minimum supporting constituent graph G_i satisfies $|V_j| < \sum_{u \in V_i} |A(u)|$ and $\sum_{v \in V_j} |A(v)| > |V_i|$ $(j \neq i)$. Then, the optimum survivability becomes $|V_i^{\text{out}}|$.

Proof. The maximum survivability achievable by restructuring is equal to the number of nodes that have at least one outgoing arc $|V_i^{\text{out}}|$ in the minimum supporting constituent graph $G_i = (V_i, E_{ii})$, because the removal of such nodes from G_i must destroy all the cycles between G_i and another constituent graph. In order to achieve the maximum survivability via the restructuring process, it is necessary that each node $u \in V_i^{\text{out}}$ belongs to a cycle whose length is 2. Otherwise, the cycle contains another node $w \in V_i^{\text{out}}$, and the removals of such w's make u lose all incoming arcs. Note that a node in $V_i \setminus V_i^{\text{out}}$ is never a part of directed cycles, since it has no outgoing arc.

Suppose that we have the minimum supporting constituent graph G_i and another constituent graph G_j that satisfy the two conditions in the lemma. From the definition of the minimum supporting constituent graph, we can make $|V_i^{\text{out}}|$ pairs of nodes $\langle u \in V_i^{\text{out}}, v \in V_j^{\text{out}} \rangle$, which are expected to form a length-2 cycle together after restructuring, so that no two nodes in V_i are paired with the same node in V_j^{out}.

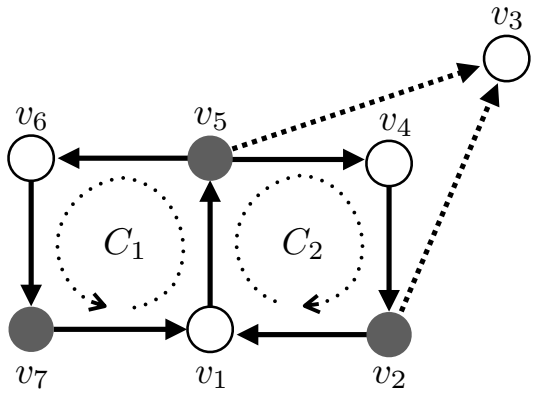

Figure 2.9. Original graph G with Marginal Arcs (v_2, v_3) and (v_5, v_3).

Figures 2.7 and 2.8 illustrate a general example of a restructuring process to form such a length-2 cycle by dependency arc relocations. Note that the figures only show A_{ji}, but the symmetric argument can be done for A_{ij}. Let $\langle u' \in V_i^{\text{out}}, v' \in V_j^{\text{out}} \rangle$ be a pair such that $(v', u') \notin A_{ji}$. In order to make a length-2 cycle between v' and u', the arc (v', u''') should be relocated to (v', u'). However, the relocation makes u''' lose all of its incoming arc. The loss of incoming arc of u''' is always avoided by relocating one of the arcs incoming to u'' to u''' (See Figures 2.7 and 2.8 (1)). The supposition in the lemma and the pigeonhole principle suggest the existence of at least one node $u'' \in V_i$ that has two incoming arcs. After the adjustment of the provisioning for u''' by this relocation, the arc (v', u''') can be relocated to (v', u') (See 2.7 and 2.8 (2)).

For a pair $\langle u' \in V_i^{\text{out}}, v' \in V_j^{\text{out}} \rangle$ such that $(u', v') \in A_{ij}$, similar relocations are always possible, because $|V_j| < \sum_{u \in V_i} |A(u)|$. Thus, these relocations eventually achieve the maximum survivability by forming $|V_i^{\text{out}}|$ length-2 cycles that each consist of a pair $\langle u \in V_i^{\text{out}}, v \in V_j^{\text{out}} \rangle$. □

Some propositions similar to Lemma 2 appear in related literature [84, 10]. The sufficient condition provided in Lemma 2 allows the entire restructuring of inter-arcs by repeated relocations of each arc. Therefore, the ΔH problem is recognized as a design problem of an entire interdependent network discussed in [84] under these assumptions. Also, the work [10]

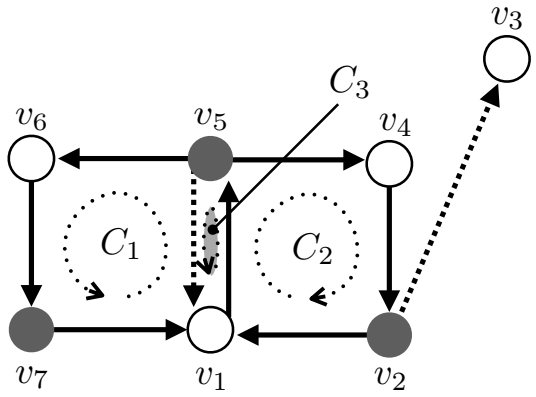

Figure 2.10. Modified graph G' with a new arc (v_5, v_1).

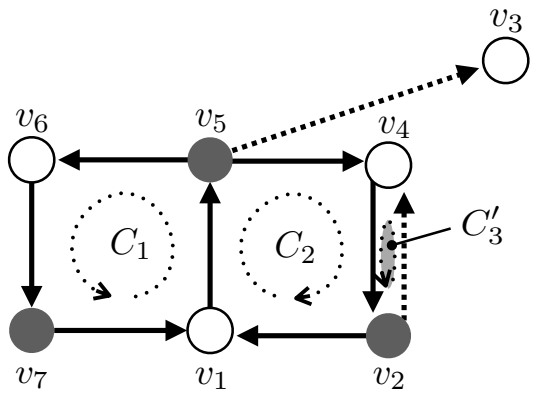

Figure 2.11. Modified graph G'' with a new arc (v_2, v_4).

claims that such a one-to-one provisioning relation realizes the robustness, while assuming certain structural characteristics of random graphs.

However, it is unrealistic to relocate a dependency arc many times, when considering the overhead of the changes of provisioning relations in network systems. Therefore, the following part of our paper discusses the case where the number of relocations are strictly restricted: $\rho((u,\cdot)_m) \leq 1$ $(1 \leq m \leq \deg_{\mathrm{out}}(u),\ \forall u \in V)$. Under this condition, it cannot be guaranteed to obtain the optimum survivability even when the sufficient condition above holds.

2.5 heuristic algorithm for ΔH Problem

This section proposes a heuristic algorithm for the clustered ΔH problem. Before providing the details of our heuristic algorithm, we first define special types of arcs named Marginal Arcs (MAs), which are candidates for the relocations in Section 2.5.1. Then, the heuristic algorithm, which consists of two algorithms: Find-MAs and ΔH, is described. The Find-MAs algorithm enumerates all the arcs that match the definition of MAs. With the set of MAs found by the Find-MAs algorithm, the ΔH algorithm decides appropriate relocations of the dependency arcs in the set, considering the disjointness of newly formed cycles, so that it can improve the survivability of a given network.

After the discussion for a simple case with only one cluster in each constituent graph in Sections 2.5.2 to 2.5.3, Section 2.5.4 explains how the other cases with multiple clusters are broken down into the simple case.

2.5.1 Restructuring of Dependencies

To guarantee continuous availability, it is necessary to classify the dependency arcs into either changeable or fixed arcs. However, it is computationally difficult to know the classification beforehand under the condition of $\rho((u, \cdot)_m) \leq 1$ $(\forall u \in V)$, because this process involves enumeration of all the permutations of arc relocations and their combinations of destinations. Thus, in this paper, the classification is simplified by using a sufficient condition, while this enumeration is likely to become another optimization problem for further investigation.

As observed in Section 2.3.3, increasing disjoint cycles in a given network could be an important factor to enhance overall survivability. Hence, our method maintains all existing cycles, which is sufficient to avoid cascading failures, and tries to reallocate the destinations of the arcs that do not belong to directed cycles and that do not make their descendant nodes nonfunctional. Let the arcs that are not in any cycles in a given directed graph $G = (V, A)$

be called *Marginal Arcs* (MAs). Formally, the set $M \subsetneq A$ of MAs is defined as

$$M := \{(u, v) \mid (u, v) \notin A(C)\ \forall C \in \mathcal{C}(G)\}. \tag{2.1}$$

Lemma 3. A removal of any marginal arc never decreases the survivability of an interdependent network: $|H(G)| \leq |H(\overline{G})|$, where G is a given graph, and $\overline{G}$ is the graph obtained by the removal.

Proof. Let M be a set of marginal arcs. From the definition of MAs (Eq. (2.1)), the removal of MAs does not destroy or connect any existing cycles in $G = (V, A)$. Therefore, $|H(G)| = |H(\overline{G})|$, where $\overline{G} = (V, A \setminus M)$. □

Moreover, appropriate relocations of the removed MAs could improve the survivability of interdependent networks, assuring operability during the relocation process and maintaining the provisioning capability of each node. Let us analyze the effect of dependency relocations using simple examples in Figures 2.9-2.11. The given graph G in Figure 2.9 has two marginal arcs: $M = \{(v_2, v_3), (v_5, v_3)\}$. In order to maintain at least one supporting node for v_3, one of the MAs has to remain the same, and the other can be relocated. Figure 2.10 shows the case of relocating (v_5, v_3) to (v_5, v_1); on the other hand, Figure 2.11 indicates the case of relocation of (v_2, v_3) to (v_2, v_4). Even though one new cycle (C_3 and C_3' respectively) is formed by each relocation, the modified graphs G' and G'' have different survivability: $|H(G')| = 1$ $(= H(G))$, and $|H(G'')| = 2$. This is because the cycles in G' are not disjoint with each other: $V(C_1) \cap V(C_2) \cap V(C_3') \neq \varnothing$; in contrast, $V(C_1) \cap V(C_2) \cap V(C_3'') = \varnothing$ in G''. Therefore, it could be said that the appropriate relocation for improving survivability is to form disjoint cycles.

2.5.2 Find-MAs Algorithm

The Find-MAs algorithm first distinguishes MAs M, which are candidate arcs for relocations, from the arcs in directed cycles in a given graph $G = (V, A)$, by employing Johnson's

Algorithm 1 ΔH-algorithm(G, l)

Input: subgraph (directed graph) $G = (V, A)$, maximum hop $l \in \mathbb{N}$ (odd)

1: $M \leftarrow$ find-MAs(G) $\quad$ # $M \subset A$
2: **for** each $(v, w) \in M$ **do**
3: **if** $\deg_{\text{in}}(w) \geq 1$ after $A \setminus \{(v, w)\}$ **then**
4: **while** True **do**
5: pick $C \in \mathcal{C}(v)$ (randomly)
6: **for** $i \leftarrow l$; $i > 0$; $i \leftarrow i - 2$ **do**
7: pick $u \in V(C) : \overline{d_C}(v, u) = i$
8: **if** $u \notin U$ **then**
9: $A \leftarrow A \setminus (v, w) \cup (v, u)$
10: $U \leftarrow U \cup \{n \mid \overline{d_C}(v, n) \leq i\}$
11: break to next arc in M (line 2)
12: **end if**
13: **end for**
14: **end while**
15: pick $(u, v) \in A_{\text{in}}(v)$ (randomly) # Minimal-add process (line 15,16)
16: $A \leftarrow A \setminus (v, w) \cup (v, u)$
17: **end if**
18: **end for**

algorithm [31]. Johnson's algorithm enumerates all elementary cycles in a directed graph within $O((|V| + |E|)(|\mathcal{C}(G)| + 1))$. It is enough for distinguishing MAs to obtain elementary directed cycles because any non-elementary cycle can be divided into multiple elementary cycles within which dependency relationships are closed. After the enumeration of cycles in G by Johnson's algorithm, the set of MAs is obtained by $M \leftarrow A \setminus \bigcup_{C \in \mathcal{C}(G)} A(C)$.

2.5.3 ΔH Algorithm

With the set of MAs obtained by Johnson's algorithm, the ΔH algorithm (shown as pseudo code in Algorithm 1) relocates the destinations of MAs, considering the disjointness of newly created cycles. (See the discussion in Section 2.5.1.) For each MA (v, w), our algorithm first checks whether or not the relocation of this MA causes the loss of supports for the current destination w: $\deg_{\text{in}}(w)_{\overline{G}=(V, A \setminus \{(v,w)\})} \geq 1$ (line 3).

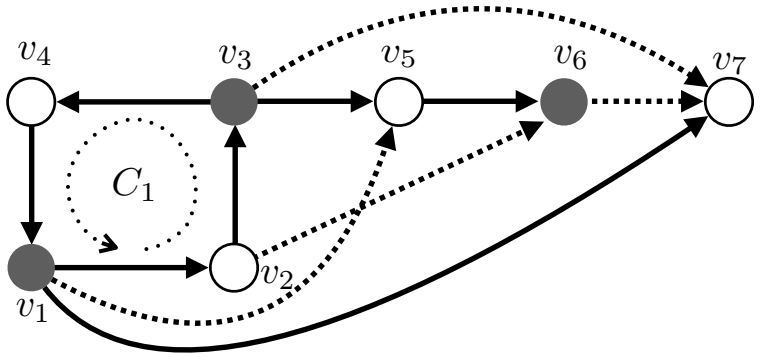

Figure 2.12. A given graph G with $M = \{(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_6, v_7), (v_1, v_7), (v_3, v_5), (v_5, v_6)\}$.

If w still has some supporting node after the removal of (v, w), the next step is determining a new destination for $(v, \cdot)$. Our algorithm randomly selects one of the cycles that contains the source v denoted by $C \in \mathcal{C}(v)$ (line 5). There may be multiple possible candidate nodes for a new destination in the cycle C. Thus, the new destination is decided by the size of the newly formed cycle, which is a result of the relocation (lines 6, 7). To represent the size of the newly formed cycle, the distance from a node v to a node u in an (existing) cycle C in the counter direction is denoted as $\overline{d_C}(v, u)$ in our pseudo code. When the maximum hop is designated by l, the algorithm tries to make a new cycle with size $l + 1$ using a node u, such that $\overline{d_C}(v, u) = l$, as the destination of the MA. If it fails to form the cycle, it attempts to compose a smaller cycle using a node u' such that $\overline{d_C}(v, u') = l - 2$. Because of the definition of the dependency, an arc must span between two different layers or constituent networks. Since the node at $\overline{d_C}(v, u) = l - 1$ in C is in the same constituent network as the source node v, it cannot be a new destination.

Consider an example using a given graph G shown in Figure 2.5 and the restructured graph in Figure 2.6. Since the removal of (v_1, v_9) does not make v_9 lose all its incoming dependency arcs, our algorithm tries to relocate the destination of this arc to one of the nodes in the cycle C_1, which are v_2, v_6, v_8. For instance, in the case $l = 3$, a new cycle C_3 is

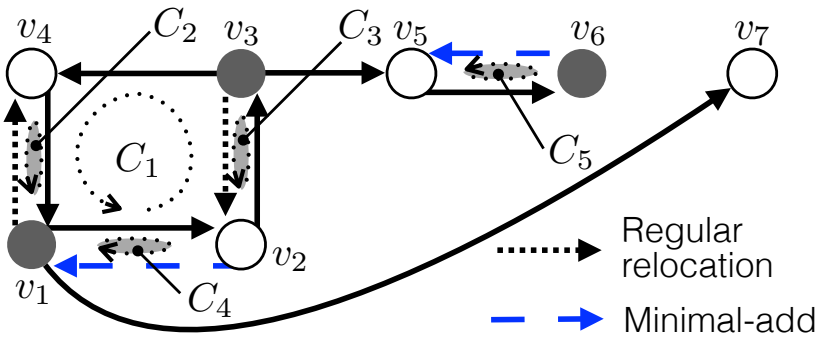

Figure 2.13. A modified graph G' with new arcs: $(v_1, v_4), (v_3, v_2), (v_2, v_1)$, (v_6, v_5).

formed as depicted in Figure 2.6 by choosing v_6, that satisfies $\overline{d_{C_1}}(v_1, v_6) = l\,(= 3)$. Similarly, if l is initialized to 1, a new cycle C_3 is formed using $\{v_1, v_8\}$.

After selecting a destination candidate u in line 7, our algorithm checks if u is already used to create a new cycle (line 8). This is confirmed by a set of nodes U storing all the nodes that are in newly formed cycles: $\{n \mid \overline{d_C}(v, n) \leq i\}$ (line 10). For instance in Figure 2.6, $U \leftarrow U \cup \{v_1, v_6, v_7, v_8\}$. As will be understood, when another MA tries to form a new cycle using one of these nodes in U, the new cycle and C_3 share some nodes, which means that those cycles are not disjoint. Also, the arc set A is updated when the new destination is finally fixed (line 9).

If there exists no possible destination for an MA (v, w) that satisfies all the conditions, the relocation of the MA is conducted by randomly selecting an incoming arc of v, (u, v) and relocating (v, w) to (v, u), so that it composes a cycle of length 2 (line 15, 16). This random selection is named *Minimal-add* process.

The MAs relocated by the Minimal-add process satisfy either of the following cases: 1) The node v does not belong to any cycles: $\mathcal{C}(v) = \varnothing$, or 2) all the nodes in the cycles of $\mathcal{C}(v)$ are already used to compose new cycles by other MAs. Figures 2.12 and 2.13 show examples of these two conditions (dashed arcs). A given graph G has the MA set $M = \{(v_1, v_5), (v_2, v_6), (v_3, v_7), (v_6, v_7),\ (v_1, v_7), (v_3, v_5), (v_5, v_6)\}$. Eventually, the ΔH algorithm respectively relocates (v_1, v_5) and (v_3, v_7) to (v_1, v_4) and (v_3, v_2). Because v_6 is not in any

cycles in G (reason 1), the Minimal-add process picks the source of one of the current incoming arcs in $A_{\text{in}}(v)$, v_5 as the new destination. Also, (v_2, v_6) does not have any possible destinations that are not in the set U (reason 2), and it is relocated to (v_2, v_1) by the Minimal-add process.

2.5.4 Application to Clustered Networks

Our heuristic algorithm employs another algorithm named *Decompose-cluster* to form subgraphs, which indicate candidate destinations for the MAs in each cluster, from a given interdependent network. When interdependent networks are clustered, the modification of the destinations of MAs needs to be conducted under more constraints given by supportability functions σ_{ij}: $\kappa_j(v \in V_j) \in \sigma_{ij}(u \in V_i)\ \forall (u, v) \in A$. The Decompose-cluster algorithm selects each cluster (node set W_i^x $(1 \leq i \leq k,\ 1 \leq x \leq \gamma_i)$) and collects MAs (u, v) whose sources are in the cluster $(u \in W_i^x)$, or whose destinations and sources are respectively in the cluster W_i^x and in a cluster in $\sigma_{ij}(v)$ $(v \in W_i^x\ \&\ \kappa_j(u \in V_j) \in \sigma_{ij}(v)))$. Using the collected MAs and their endpoints, a subgraph Y for reallocations of MAs in W_i^x is composed. Each subgraph for each cluster is given to the ΔH-algorithm so that it can improve the survivability by restructuring dependencies in the subgraph.

As will be understood, no directed cycles exist if no MA matches the condition of $v \in W_i^x$ $\&\ \kappa_j(u \in V_j) \in \sigma_{ij}(v)$. However, this is not going to happen in our work due to the assumption mentioned in Section 2.4.1. Note that the absence of such MAs means that nodes in a cluster x are not provided any support by the nodes that receive some supports from the nodes in the cluster x.

2.5.5 Complexity Analysis

The Decompose-cluster algorithm extracts $\sum_{i=1}^{k} \gamma_i$ subgraphs from a given graph $G = (V, A)$. The number of clusters γ_i in each constituent graph tends to be much smaller than the

Algorithm 2 Decompose-cluster(G)

Input: interdependent network (directed graph) $G = (V = \bigcup_{i=1}^{k} V_i, A)$,
clustering functions κ_i

```
D ← ∅
for a node set W_i^x (1 ≤ i ≤ k, 1 ≤ x ≤ γ_i) do
    P ← ∅, R ← ∅
    for each (u, v) ∈ A \ D do
        if u ∈ W_i^x or (v ∈ W_i^x & κ_j(u ∈ V_j) ∈ σ_ij(v)) then
            P ← P ∪ {u, v}
            R ← R ∪ (u, v)
            D ← D ∪ (u, v)
        end if
    end for
    compose graph Y = (P, R)
    ΔH-algorithm(Y, l)
end for
```

number of nodes; thus, $\sum_{i=1}^{k} \gamma_i$ can be considered as a constant. In order to compose each subgraph, the algorithm requires checking the source and destination of each arc in A. However, each edge appears in exactly one subgraph because of the used edge set D. Therefore, the total complexity of the Decompose-cluster algorithm is $O(|V| + |A|)$.

The complexity of the ΔH-algorithm is sensitive to the number of cycles in the interdependent network. It is known that Johnson's algorithm finds all elementary cycles within $O((|V| + |E|)(|\mathcal{C}(G)| + 1))$. The ΔH-algorithm determines a new destination after $\frac{l}{2} \times \mathcal{C}(G)$ searches for each MA, in the worst case. When only one cycle whose size is 2 exists in the input, and the other nodes are supported by the cycle, the size of the set M becomes $|E| - 2$. It is obvious that the complexity of the Minimal-add process is $O(1)$, so the worst case analysis takes the case where all MAs are reallocated by the ΔH-algorithm. Thus, its complexity is $O((|V|+|E|)(|\mathcal{C}(G)|+1))+O((|E|-2)(\lceil \frac{l}{2} \rceil \times |\mathcal{C}(G)|))$. Assuming the maximum hop l is small enough to be considered as a constant, the overall complexity of our heuristic algorithm becomes $O((|V| + |E|)|\mathcal{C}(G)|)$. Note that the assumption on l is valid with our strategy, which tries to increase disjoint directed cycles in a given graph.

2.5.6 Optimality in Special Graphs

To analyze the performance of our heuristic algorithm, we consider the survivability improvement in special graphs where either an exhaustive search gives us the optimum survivability, or some special properties allow us to compute the optimum.

In the analysis, the upper bound of the survivability improvement, which is used as a benchmark for the rest of this paper, is calculated based on the number of the MAs that satisfy the following two conditions. First, let V_s be a set of nodes that hold more than one MA, and M_s be a set of MAs whose source nodes are in V_s. Even when the MAs from $v \in V_s$ form more than one new cycle, the removal of such a source node v can destroy all the newly formed cycles. This indicates that restructuring increases the survivability by at most $|V_s|$, when relocating MAs in M_s. Second, let V_d be a set of nodes whose incoming arcs are all MAs, and M_d be a set of MAs whose destination nodes are in V_d. If all the MAs incident to $v \in V_d$ are relocated, v loses its functionality during this restructuring. Therefore, at least one MA should remain as an incoming arc to v. This implies that the number of cycles newly formed by the MAs in M_d is at most $|M_d| - |V_d|$. Thus, the upper bound U is obtained by $|M| - |M_s| + |V_s| - |V_d|$.

Figure 2.14 illustrates a comparison of our algorithm with the optimum solution in a small interdependent network such that each constituent graph has 15 nodes, and the number of dependency arcs is 84, including 5 MAs. The optimum solution is obtained by an exhaustive search of 759,375 combinations of reallocations. This numerical example shows that the solution given by the ΔH algorithm would not provide solutions that are exceptionally divergent from the optimum solution. It also infers that the upper bound is not tight in general.

Figure 2.15 indicates that the survivability obtained by our restructuring heuristic algorithm matches the optimum in a special class of graphs, which are named MA-saturated

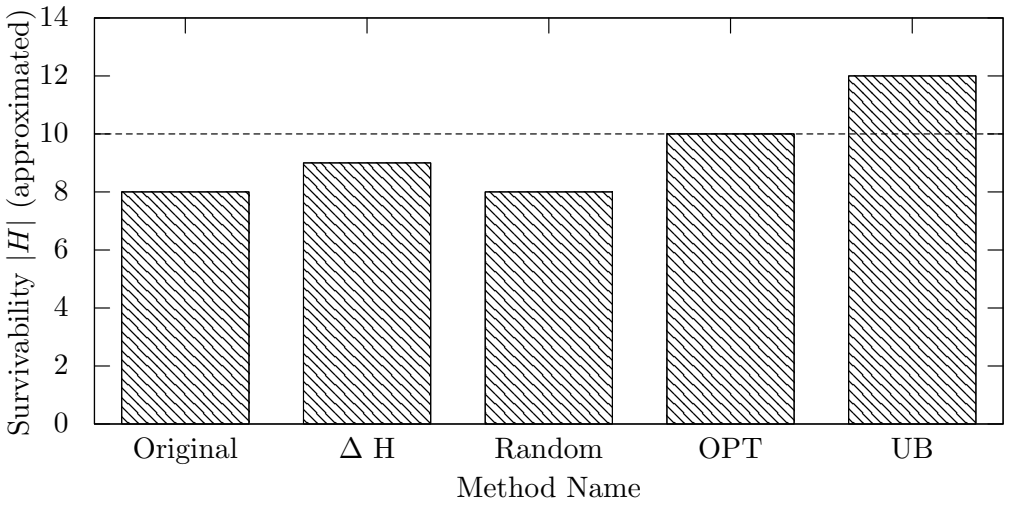

Figure 2.14. Numerical comparison with the optimum solution in a small interdependent network.

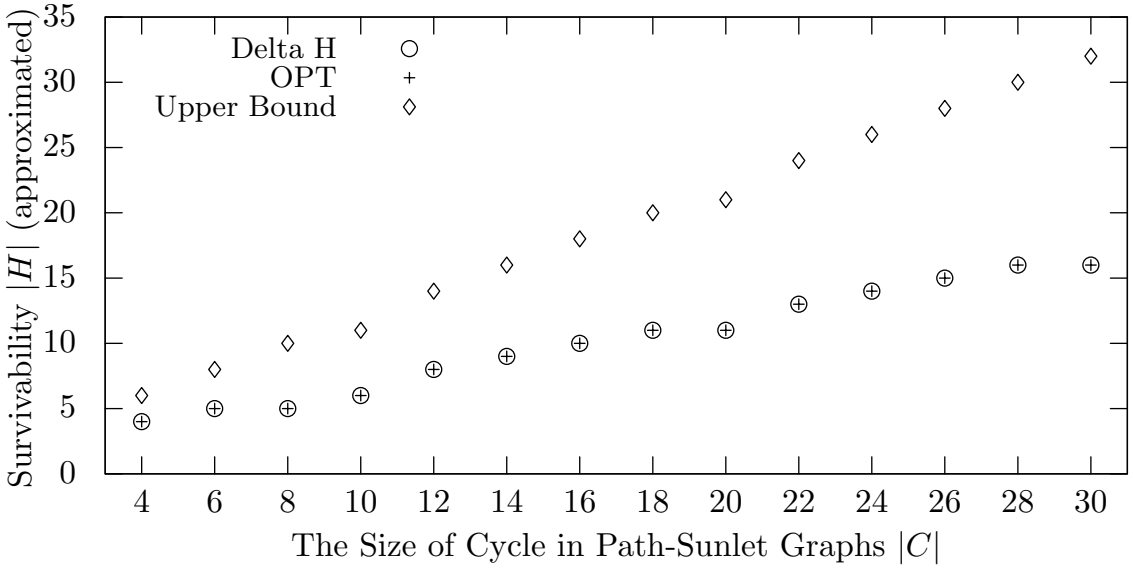

Figure 2.15. Survivability of MA-saturated Path-Sunlet graphs $\zeta_2(G \in \mathcal{L})$ with two length-3 paths: $|\mathcal{P}| = 2$, $k_i = 3$ $(\forall P_i \in \mathcal{P})$.

Path-Sunlet Graphs $\zeta_2(G),\ G \in \mathcal{S}$. The optimum value of survivability for these graphs is always computable based on the following discussion.

Definition 1. Path-Sunlet Graphs $\mathcal{L}$: A set of graphs satisfying the following conditions are named Path-Sunlet graphs. Let $\mathcal{L}$ denote the set of Path-Sunlet graphs.

- $G \in \mathcal{L}$ only has one cycle C.
- The arcs that are not in the cycle C form a set of disjoint paths whose initial nodes are in C: $\mathcal{P} = \{P_i = (v_1^i, v_2^i, ..., v_{k_i}^i) \mid v_1^i \in C \text{ and } P_i \cap P_j = \varnothing \ (\forall P_{j \neq i} \in \mathcal{P})\}$.

Definition 2. MA-saturation $\zeta_\delta(G)$ of a graph G: The MA-saturation is an operation of adding additional arcs to a given graph until any addition of an arc makes the graph non-simple, maintaining the out-degree constraint that the out-degree of any node does not exceed a given constant $\delta \in \mathbb{N}$.

Claim 1. The optimal restructuring of MAs in MA-saturated Path-Sunlet graphs $\zeta_2(G),\ G \in \mathcal{L}$ consists of forming length-2 cycles using an MA and an edge in either $P_i \in \mathcal{P}$ or C.

We consider the cases where $|\mathcal{P}| \geq 1$, because the survivability in the case of $|\mathcal{P}| = 0$ is obviously $\left\lceil \frac{|V(C)|}{2} \right\rceil$.

Lemma 4. By removing arcs that are not in any cycle, the optimally restructured MA-saturated Path-Sunlet graph $\zeta_2(G)$ is decomposed into some sequence of cycles.

Proof. Three or more cycles do not meet at the same node, since $\delta = 2$. Therefore, the only possible topology with multiple length-2 cycles is a chain of cycles, in which two cycles share exactly one node. □

Lemma 5. The survivability of the optimally restructured MA-saturated Path-Sunlet graphs $\zeta_2(G),\ G \in \mathcal{L}$ is $\sum_{q \in Q} \left\lceil \frac{q}{2} \right\rceil$, where Q is the set of all the sequences of cycles obtained by removing the arcs that are not in any cycles.

Proof. Removal of one node that is shared by two cycles breaks the two cycles. When q is even, the process gives us the survivability of $\frac{q}{2}$. If q is odd, one additional removal is needed to destroy the remaining cycle. Thus, the survivability of a sequence of q cycles is $\left\lceil \frac{q}{2} \right\rceil$.

Since each sequence in Q is disjoint with the other, the survivability of the entire graph is obtained by summing up the survivability of each sequence. □

2.6 Simulation

To understand the performance of the proposed algorithm, our simulations are conducted in both non-clustered and clustered interdependent network models of different sizes. The results from the simplest cases where each constituent network only consists of one cluster (non-clustered) are first described, and the clustered cases follow.

2.6.1 Network Topology

The performance of the proposed algorithm is analyzed in random directed bipartite graphs that contain at least one directed cycle. Assuming the situation in which a current interdependent network is working normally, each node is either a member of some cycle or reachable from a node in a cycle through some directed path in the input graph. Because our algorithm only concerns the dependency arcs between 2 constituent graphs ($k = 2$), any interdependent network is represented as a directed bipartite graph whose arcs connect a pair of different types of nodes.

Each random bipartite graph is generated by specifying the following parameter: V_i, $\max_{v \in V} \deg_{\text{in}}(v)$ and $\min_{v \in V} \deg_{\text{in}}(v)$. In order to observe the performance in different conditions, experiments are conducted in symmetric and asymmetric interdependent networks. A symmetric interdependent network has constituent networks which each have identical number of nodes: $|V_1| = |V_2|$, while constituent networks of an asymmetric interdependent network have different number of nodes: $|V_1| = \frac{|V_2|}{q}$ ($q \in \mathbb{N}$). The degree of each node is determined based on the uniform distribution between the given maximum and minimum incoming degree.

2.6.2 Clustering Settings

As the non-clustered cases have symmetric and asymmetric constituent graphs, clustered interdependent networks are also examined in three patterns of topology configurations. In

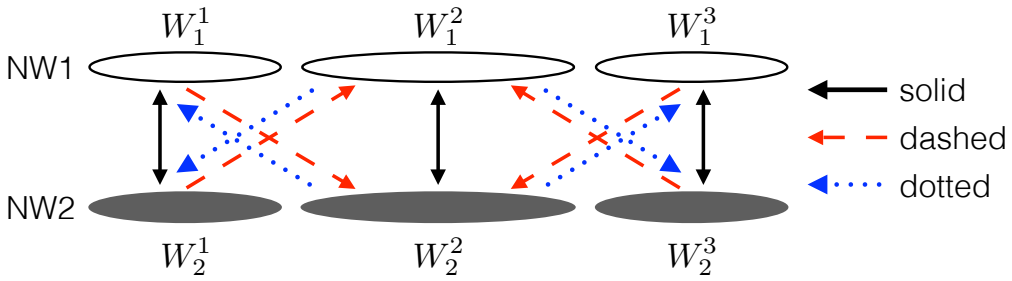

Figure 2.16. Dependency models of clustered interdependent networks. Arrows show the dependency relationships between clusters. Model 1: solid. Model 2: solid and dashed. Model 3: solid, dashed, and dotted.

our simulations, each constituent graph has three clusters: W_i^1, W_i^2 and W_i^3 $(i = 1, 2)$ (See Figure 2.16). In symmetric cases, a pair of corresponding clusters in different constituent graphs have the same number of nodes: $W_1^x = W_2^x$, while a cluster is half-sized to the corresponding cluster in the other constituent graph in asymmetric models: $W_1^x = \frac{W_2^x}{2}$.

Also, Figure 2.16 illustrates the three models that have different dependency relationships indicated as arrows. Note that when an arrow is drawn from W_i^x to $W_j^{x'}$, it means that the nodes in cluster $W_j^{x'}$ can have supports from the nodes in W_i^x. Model 1 consists only of the solid arrows, which means that each pair of corresponding clusters has dependency relationships. Model 2 has the dependencies illustrated by the solid and dashed arrows, while Model 3 has all the arrows (solid, dashed, and dotted). A major difference between these models is the possibility for a network to have some directed cycles over three or more clusters. In Models 1 and 2, directed cycles can exist only in a subgraph consisting of W_1^1 and W_2^1, W_1^2 and W_2^2, or W_1^3 and W_2^3, while a directed cycle can lie over the entire graph containing all the clusters in Model 3.

2.6.3 Metrics

The survivability of the given graphs, restructured graphs, randomly reassigned graphs, and the upper bound of the improvement are illustrated in our results. The random reassignments

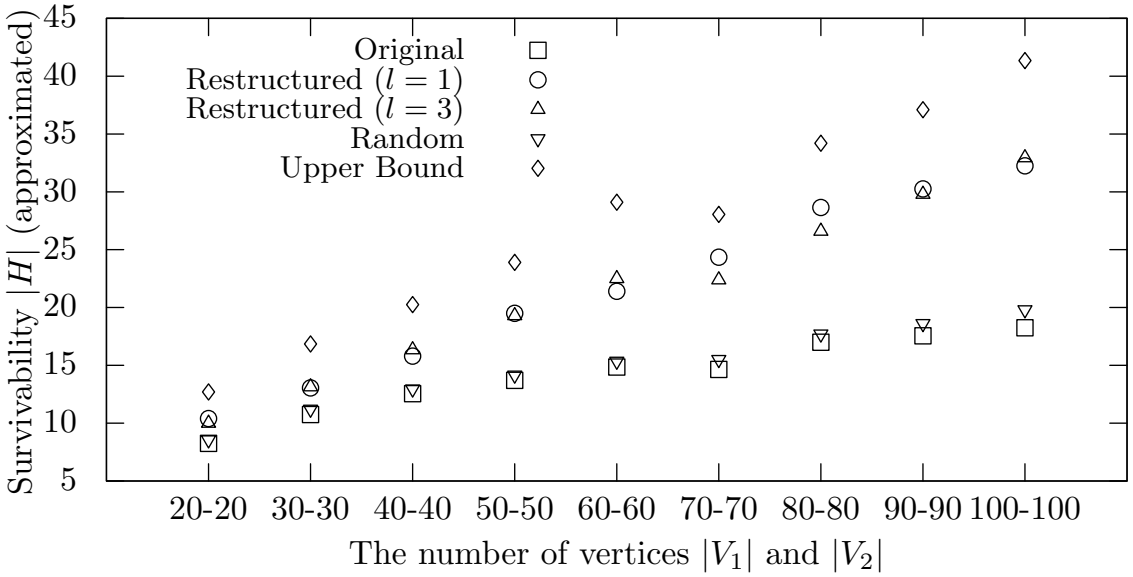

Figure 2.17. Survivability of interdependent networks before and after the improvement under $|V_1| = |V_2|$, $\max_{v \in V} \deg_{\text{in}}(v) = 4$, and $\min_{v \in V} \deg_{\text{in}}(v) = 2$, and $l = 1, 3$.

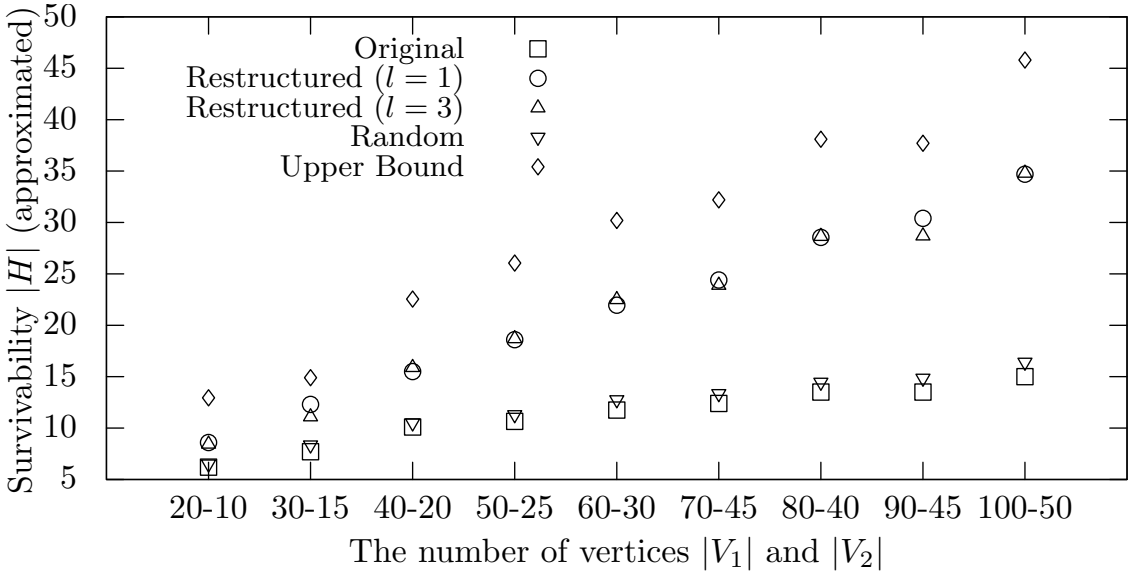

Figure 2.18. Survivability of interdependent networks before and after the improvement under $|V_1| = \frac{|V_2|}{2}$, $\max_{v \in V} \deg_{\text{in}}(v) = 4$, $\min_{v \in V} \deg_{\text{in}}(v) = 2$, and $l = 1, 3$.

of MAs are conducted with a uniform distribution over all the nodes in the other constituent graph from the constituent graph that includes the source of an MA.

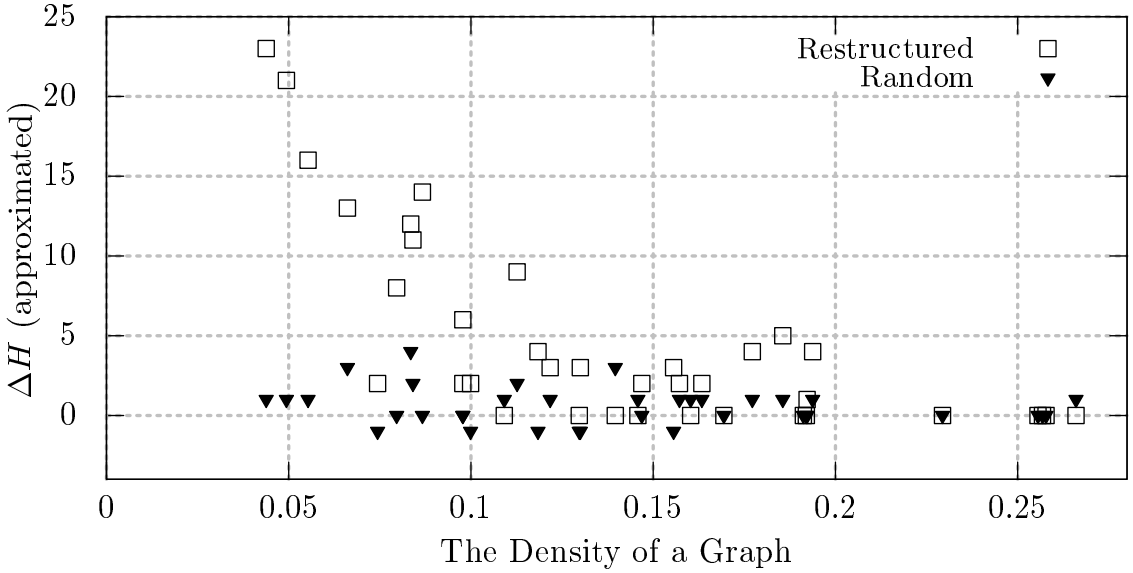

Figure 2.19. The relationship between graph density and ΔH.

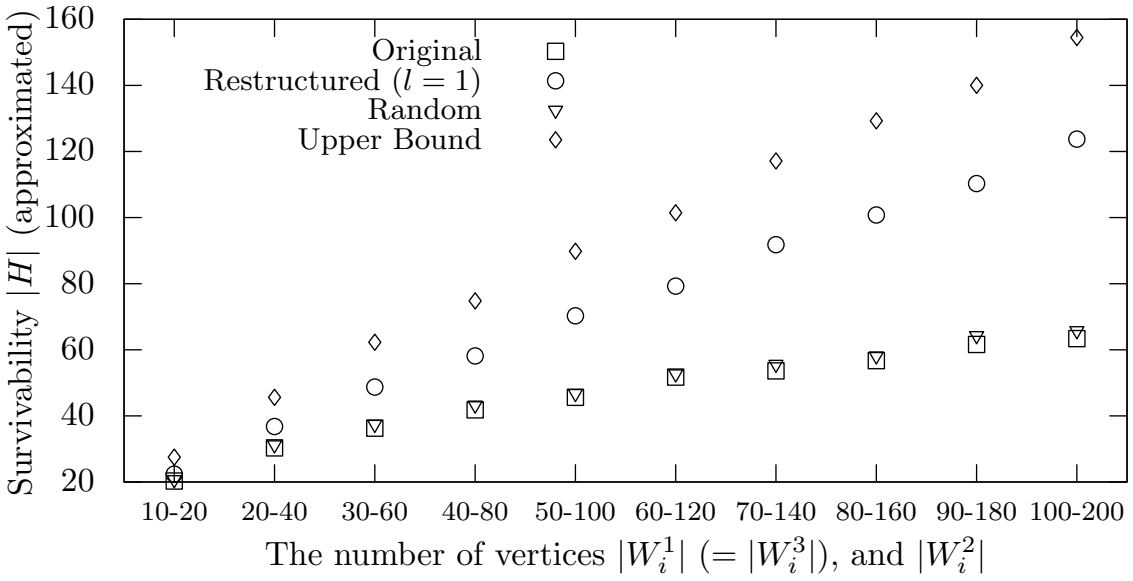

Figure 2.20. Survivability of clustered interdependent networks (Model 2) before/after the improvement under $|W_1^1| = |W_1^3| = |W_2^1| = |W_2^3|$, $|W_1^2| = |W_2^2|$, $\max_{v \in V} \deg_{\text{in}}(v) = 4$, $\min_{v \in V} \deg_{\text{in}}(v) = 2$, and $l = 1$.

Computing the size of the cycle hitting set is known to be NP-complete even in bipartite graphs, so the exact value cannot be obtained in larger graphs. Our evaluation is conducted using a well-known approximation algorithm whose approximation factor is $\ln |V| + 1$ [12].

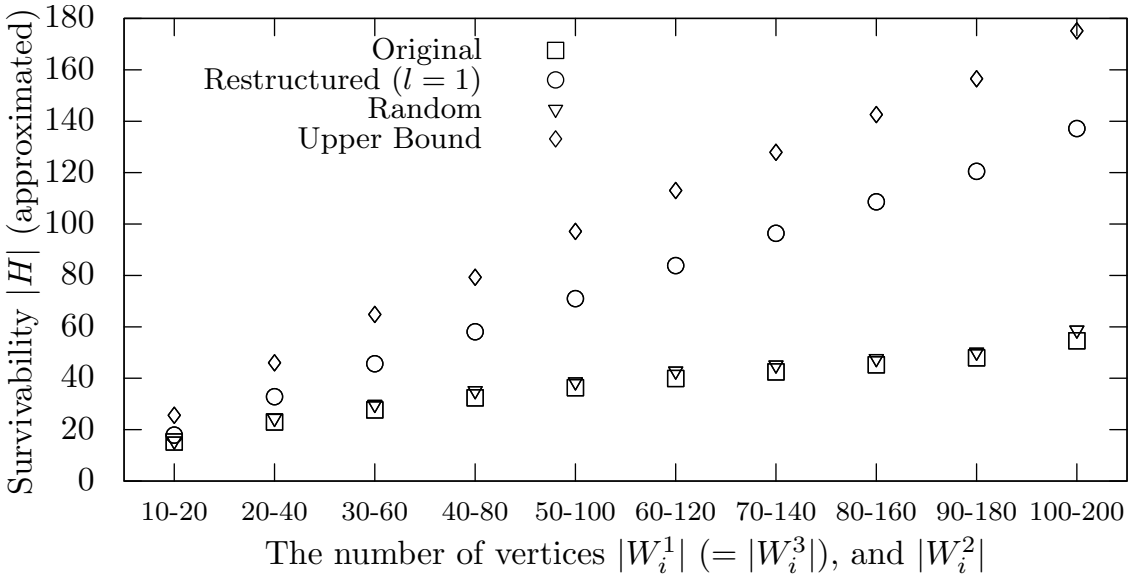

Figure 2.21. Survivability of clustered interdependent networks (Model 2) before/after the improvement under $|W_1^1| = |W_1^3| = \frac{|W_2^1|}{2} = \frac{|W_2^3|}{2}$, $|W_1^2| = \frac{|W_2^2|}{2}$, $\max_{v \in V} \deg_{\text{in}}(v) = 4$, $\min_{v \in V} \deg_{\text{in}}(v) = 2$, and $l = 1$.

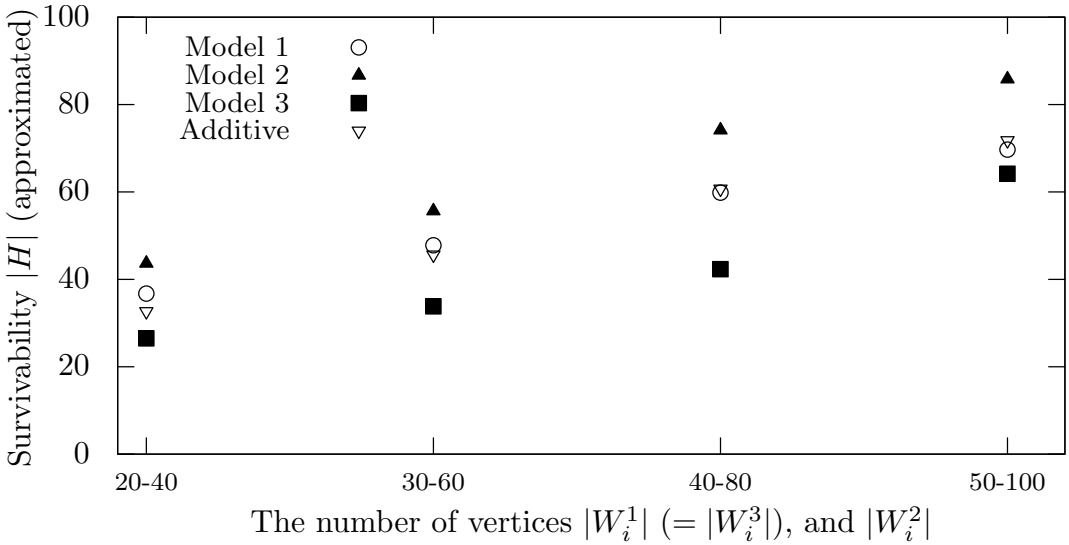

Figure 2.22. Comparison of survivability among different dependency models.

Furthermore, the density of a given graph $G = (V, A)$ defined by $\frac{|A|}{\prod_i |V_i|}$ is used to examine the relationship between the survivability improvement, and the maximum and minimum degrees.

2.6.4 Results

Non-clustered Cases

Figures 2.17 and 2.18 illustrate the survivability of the given and restructured graphs with identical and halved size constituent graphs, respectively. In both cases, our method demonstrates more improvement of the survivability compared to the random reassignment. The survivability of the original graphs $|H(G)|$ maintains a similar value regardless of the size of graphs, though the survivability of the graphs restructured by our method $|H(G')|$ steeply increases along with the size of the graph. Since, in the original graph G, arcs are randomly added, it could be difficult to form larger directed cycles. Therefore, it is reasonable that the number of disjoint cycles indicates the tendency to stay within a similar range of values. On the other hand, there would exist more MAs in larger graphs, because these graphs have more arcs that are not in directed cycles. This results in a dramatic enhancement of the survivability in larger graphs. The difference caused by the given maximum hop l for our algorithm remains small over all sizes of a graph.

Figure 2.19 indicates the relationship between the density of graphs and ΔH, the amount of survivability improvement. We compare our method to the random reassignment. The result shows that, in graphs with lower density, our method has greater success in increasing the survivability. An observed general trend of our method is the gradual decrease in ΔH in accordance with the density. This trend seems to be induced by the fact that the graphs with more arcs have a higher possibility of composing cycles even in the original topology. This implies that graphs with higher density have fewer MAs that can form new disjoint cycles. On the other hand, the random reassignment does not demonstrate its effectiveness for the improvement in graphs with any density, which is the same result from Figures 2.17 and 2.18. Moreover, the random reassignment sometimes decreases the survivability ($\Delta H < 0$). It is conceivable that the reassignment connects two (or more) cycles and makes it possible

to decompose all these cycles by the removal of a node. This result implies that imprudent restructuring of the dependency may cause more fragility of the interdependent networks.

Clustered Cases

The results in clustered interdependent networks whose dependency relationships follow Model 2 are shown in Figures 2.20 and 2.21. Similar trends to non-clustered cases are observed for both symmetric and asymmetric cases. The proposed method succeeds in increasing the survivability for different sizes of interdependent networks.

Figure 2.22 illustrates the difference in survivability after restructuring among the three types of dependency models of symmetric networks. The value of "Additive" is obtained by the simple addition of non-clustered cases that jointly compose a clustered case. For instance, the case of clustered networks consisting of 20, 40, and 20 nodes clusters is compared with the sum of the survivability of the cases of non-clustered networks of 20, 40, and 20 nodes shown in Figure 2.17. The dependency relations among clusters increase from Model 1 to Model 3 (See Fig 2.16).

Model 1 gives similar survivability to the simple addition of non-clustered cases, since a pair of corresponding clusters in two constituent graphs is independent from the other pairs in this model. In Model 2, the survivability of the entire network increases, because the nodes in cluster W_i^2 can have more supports from the clusters whose cycles are disjoint from the cycles in W_i^2. Although more supports exist among the clusters in Model 3, its survivability is less than the other models. In Model 3, a cycle can lie on more clusters because of the bidirectional dependencies among all the clusters. This topological characteristic is likely to increase the overlapping of multiple cycles and results in the decline of survivability in this model. These results cast a doubt on a naive statement claiming that the increase of dependencies induces more fragility in general interdependent networks.

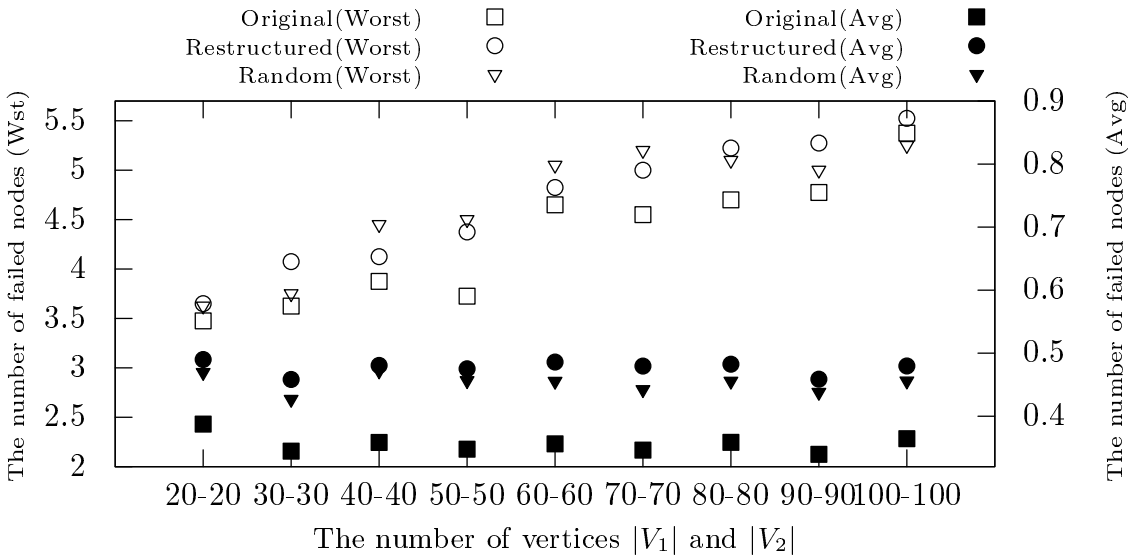

Figure 2.23. The number of failed nodes (Worst case and Average) after a single node failure under $|V_1| = |V_2|$, $\max_{v \in V} \deg_{\text{in}}(v) = 4$, $\min_{v \in V} \deg_{\text{in}}(v) = 2$, and $l = 1$.

2.7 Discussion: Impact Alleviation vs Survivability

Although it is not the primary focus of this paper, in this section, we evaluate the behavior of the proposed algorithms in terms of their effect on the size or impact of a cascading failure. Figure 2.23 illustrates the influence of our dependency modifications on the size of cascading failures induced by a single node. In this experiment, the impact of a single node failure at a node v is defined as the number of nodes θ_v that become nonfunctional after a cascading failure initiated by the failure of v. The results are analyzed in terms of the following two metrics:

- *Worst* (non-filled points): the size of the largest cascading failure: $\max_{v \in V} \theta_v$,
- *Average* (filled points): the average size of all possible cascading failures: $\frac{\sum_{v \in V} \theta_v}{|V|}$.

The robustness of restructured networks against a single node failure always declines in comparison with the original topology. The decline in the size of the largest cascading failure is most remarkable in the case of $|V_1| = |V_2| = 50$ in our simulation. In this case, the size of a cascading failure increases by 1 node after the restructuring.

In general, the concentrations of provisioning on a certain portion of a network can improve the survivability, though it can make the other portions more fragile. In contrast, appropriate distributions of provisioning are necessary to alleviate the impact of any possible single node failure. This difference in robustness against single node failures and system survivability could be a reason for the decline.

However, when examining the average size of cascading failures, it is observed that the increase in the average number of failed nodes is suppressed within 0.1 nodes over all network sizes. Thus, it could be said that our method does not deteriorate the robustness against single node failures.

2.8 Conclusion

This paper addresses the design problem of survivable clustered interdependent networks under some constraints relating to the existence of legacy systems during restructuring. Based on the definition of the survivability proposed in related work, it is claimed that the increase of disjoint cycles could enhance the survivability. The proposed heuristic algorithm tries to compose new disjoint cycles by gradual relocations of certain dependencies (Marginal Arcs) to guarantee the functionality of existing systems. Our simulations indicate that the algorithm succeeds in increasing the survivability, especially in networks with fewer dependencies. Moreover, the empirical result implies that the number of dependencies, in general, is not the root cause of the vulnerability to cascading failures. Rather, the appropriate additions of dependencies can improve the overall survivability, while poorly designed dependencies make networks more fragile. When redesigning the interdependency between control and functional entities in SDN, NFV, or CPSs based on the proposed algorithm, the possibility to experience catastrophic cascading failures would decrease.

CHAPTER 3

DEEPPR: PROGRESSIVE RECOVERY FOR INTERDEPENDENT VNFS WITH DEEP REINFORCEMENT LEARNING

The increasing demand for diverse network services entails more flexible networks that are realized by virtualized network equipment and functions. When such advanced network systems face a massive failure by natural disasters or attacks, the recovery of the entire system may be conducted progressively due to limited repair resources. The prioritization of network equipment in the recovery phase influences the interim computation and communication capability of systems since the systems are operated under partial functionality. Hence, finding the best recovery order is a critical problem, which is further complicated by virtualization due to the interdependence between Virtualized Network Functions (VNFs) and infrastructure components. This chapter[1] discusses a progressive recovery problem under limited resource availability in networks with VNFs, where some interdependencies exist. We prove the NP-hardness of the progressive recovery problem and approach the optimum solution by introducing DeepPR, a progressive recovery technique based on Deep Reinforcement Learning (Deep RL). Our simulation results indicate that DeepPR can achieve near-optimal solutions in certain networks and is more robust to adversarial failures, compared to a baseline heuristic algorithm.

---

[1]The content of this chapter is based on the following earlier works.

3.1 Introduction

Network Function Virtualization (NFV), which realizes various network functions as software components running on commodity servers, is a key enabler of the fifth generation (5G) mobile networks. To accommodate different types of requests with diverse performance requirements in the 5G era, Network Service Providers (NSPs) deliver network slices, which are sets of virtualized computing, storage, and communication equipment. Network slicing benefits each user by allocating an independent network slice carefully tailored for a specific request from the user. Network slicing is also potentially beneficial for NSPs in terms of economics and ease of configuration, compared to the case in which multiple services to different users are provided over a single uniformly designed network.

The sophisticated tailoring of resources across multiple types of network equipment demands effective orchestration that manages, monitors, and reconfigures the equipment. ETSI defines a basic framework, NFV Management and Orchestration (MANO), for this purpose [15, 14]. Figure 3.1 illustrates simplified interactions between MANO and other network components. The virtualized infrastructure (or physical) components are servers, storage, and a network. Given the state information of the infrastructure, MANO dynamically configures VNFs to provide a requested service. Usually, each VNF is designed to be a simple functional module such as a mobile content delivery network, traffic offloader, virtual EPC, or deep packet inspection (DPI) [66, 24].

The interactions among the servers, MANO, and VNFs introduce unique properties to a network system known as an *interdependent network* where two different types of nodes (i.e., VNFs and infrastructure nodes) depend on each other. This interdependency originates from the fact that MANO itself could be implemented as a special network function that runs on a commodity server with other VNFs [15]. For example, a production-quality MANO stack is available through an open-source project named Open Source MANO run by ETSI. In this paper, we refer to the VNFs and MANO as the function layer, and the physical components

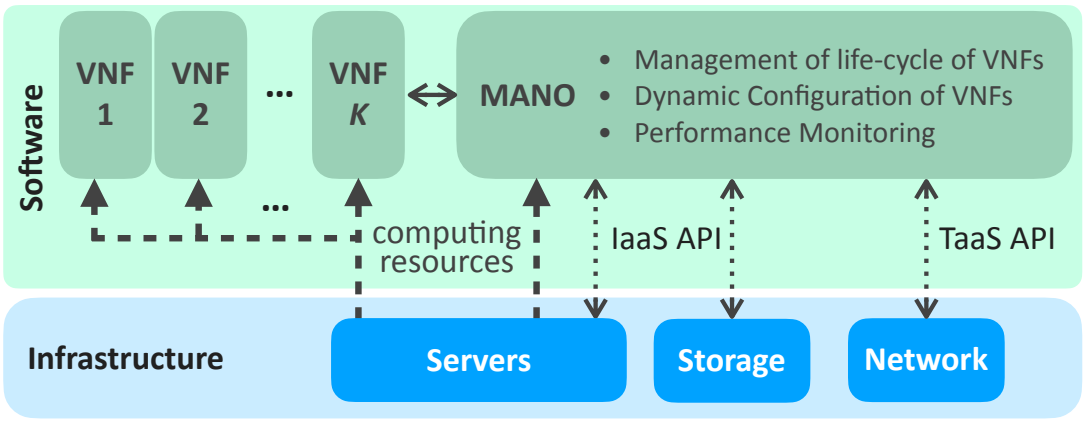

Figure 3.1. An Architecture of Service Platforms with MANO.

as the infrastructure layer. Under the assumption of MANO realized as a special VNF, all servers that are hosting other VNFs should be reachable to the virtualized MANO using a path on the infrastructure network; otherwise, they cannot collaboratively provide an appropriate network slice specified by orchestration decisions. Simultaneously, MANO can be functional only when a working server hosts it. This bidirectional dependency between MANO and infrastructure servers causes the interdependency. This interdependency model with the orchestration function is also discussed by Liu et al. in their work [39].

Although the system controlled by such orchestrators relishes flexibility in configuration, the interdependency also introduces new fragility, which potentially can induce a massive failure over the entire system and additional complexity during recovery [39, 35, 57]. The works [39, 57] show that *cascading failure* phenomena, which are chained failure events spread from the failure of some network node through the interdependency, can cause massive failures in such interdependent networks.

After massive failures, it is critical to start providing best-effort connections and services by using available repair resources such as manpower or backup equipment, which would be very limited at earlier stages of recovery. Our goal in this paper is to provide a recovery strategy to cope with such failures specifically under the limited amount of resources. The

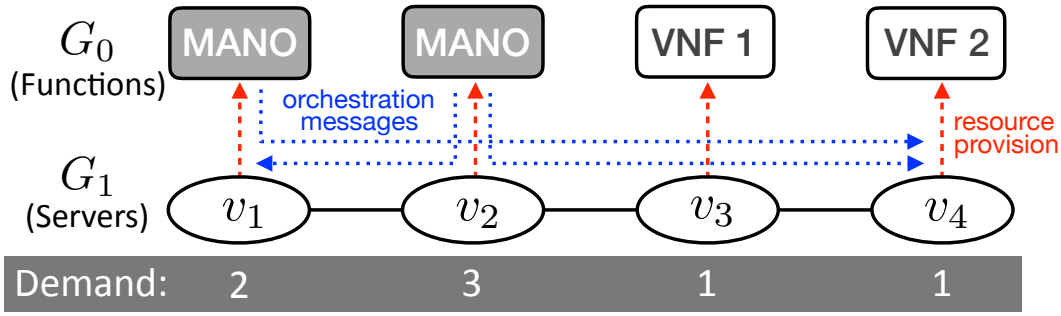

Figure 3.2. A Motivating Example: Every infrastructure node requires a connection to at least one orchestration function node in the function layer to receive orchestration messages. Also, each function node needs to be provided with computation resources by the infrastructure node hosting it.

repair resource allocation problem to decide the prioritization of equipment recovery is well-studied in [76, 53, 13] for networks without the interdependency. Also, the work in [19] considers one-directional dependency in a variation of the prioritization problem in inter-datacenter networks. However, this prioritization becomes more complex when there is interdependency between two types of nodes in a network since the role of each node is determined not only by the topology of a network but also by the interdependency [83, 41]. The following example characterizes the inherent complexity of the problem.

Let us consider an example illustrated in Figure 3.2, which simplifies the relationship between the function layer and the infrastructure layer depicted in Figure 3.1. The network consists of two types of nodes: function nodes in the function layer (G_0), and infrastructure nodes in the infrastructure layer (G_1). Suppose that each infrastructure node v_i on G_1 hosts one function node, and v_1 and v_2 have replicas of a virtualized orchestration function (MANO). The dotted arrows in Figure 3.2 illustrate the interdependency between function nodes and infrastructure nodes. Function nodes are dependent on the infrastructure nodes that host them, and infrastructure nodes are dependent on the MANO function nodes that control them.

To model a recovery problem after a major failure, two node parameters, *demand* and *utility*, are introduced. The demand of an infrastructure node represents the degree of damage

from which the node suffers. For example, each server would receive different damages from an earthquake depending on the distance from the epicenter. The utility of an infrastructure node indicates the importance of the node based on the quantified popularity of the functions that the node hosts. For instance, a node that provides VNFs that are requested by more network slices has a higher utility since the recovery of the node contributes to more users. Note that it is assumed that the system is already equipped with monitors, which have monitoring capability similar to the monitors in [13], and can estimate appropriate demands for every node.

Our problem is to determine the recovery order of the infrastructure nodes over multiple time steps while assuming that a certain amount of repair resources becomes available at each time step. The progressive property can be interpreted as a situation where manpower and physical resources gradually arrive to fix the failure as time passes from the first failure incident. Here, the following two recovery orders are compared: $P_1 : v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ and $P_2 : v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$. For simplicity, it is assumed that every infrastructure node has a utility of 1, and only one unit of resource is available at each time step t_i $(1 \leq i \leq 7)$.

Table 4.2 describes the accumulative utility at each time step when following each recovery order, P_1 and P_2. As explained above, the interdependency insists that a node must have both sufficient resources and a path to a working orchestration function (MANO) to be considered recovered. For instance, in P_1, we first recover v_1 and obtain utility of 1 at t_2 since it takes two steps to satisfy the demand of the node. A recovered node stays functional until the last step t_7 and continues providing the same utility at every step. In P_2, the interdependency plays an interesting role in the recovery process. Even though sufficient resources are assigned to v_4 and v_3 in the first two steps, the utility remains 0 until v_2 is recovered. This is because the two nodes (v_3, v_4) are not reachable from the orchestrator. Hence, the total utility jumps to 3, once v_2 is recovered at t_5. As a result, the total utility of P_1 over time is 12 while the total utility of P_2 is 10.

Table 3.1. Utility during Recovery Process: The difference in the recovery order causes loss of potential interim utility.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
P_1	0	1 (v_1)	1	1	2 (v_2)	3 (v_3)	4 (v_4)
P_2	0	0	0	0	3 (v_2, v_3, v_4)	3	4 (v_1)

Hence, the total utility available during recovery is different depending on which recovery order we adopt. Motivated by this simple example, the question addressed in this paper is the following. *How do we find a recovery order that maximizes the accumulated utility during the recovery process in networks with interdependency between the function layer and the infrastructure layer?* This problem is a variant of the progressive recovery problem [76], which aims at maximizing the total flow going through a network during the recovery process. However, the fundamental difference lies in the consideration of the interdependency between nodes in the two layers. In a network without interdependency, an infrastructure node is recovered if sufficient repair resources have been allocated to the node. However, in interdependent networks, allocating a sufficient amount of resources is not enough to recover an infrastructure node, because each infrastructure node also needs to be reachable to a working orchestration function. This reachability constraint for recovery of infrastructure nodes would suggest a certain order that balances the prioritization of the infrastructure nodes that have larger utility and the effectiveness to assign resources to the infrastructure nodes that can become functional immediately after the resource allocation.

Our major contribution is twofold: (1) a set of theoretical results, which narrow down decision-making factors during the recovery process, and (2) a Deep Reinforcement Learning-based (Deep RL) algorithm to decide the recovery order. Combined with the theoretical results that provide guidelines on the selection of a recovery order, the Deep RL technique demonstrates its performance as a general method to solve the recovery problem, which answers the research question above.

In our preceding conference paper [29], we have demonstrated the performance of our Deep RL based algorithm (DeepPR) under random failures. This journal paper provides an analysis of DeepPR under more failure scenarios, which include adversarial scenarios. In other words, we analyze the robustness of a recovery algorithm in the case where a malicious party attempts to decrease the total interim utility offered by the network. This reconsideration motivates us to integrate a promising classical heuristic into DeepPR, which further suggests a general methodology of solving intractable combinatorial optimization problems using Deep RL. Additionally, a complete view of the theoretical results, which are partially discussed in the conference version, is provided. The theoretical discussion enables us to cope with varieties of input instances of the progressive recovery problem in interdependent networks and to use DeepPR in any interdependent networks.

The following summarizes the contributions of this paper.

- We prove that the progressive recovery problem in an interdependent network always has an equivalent progressive recovery problem in a special graph called an *interdependency-embedded graph* (Section 3.5 - Theorem 4).

- The NP-hardness of the progressive recovery problem in small interdependency-embedded graphs is shown, which implies that the problem in any interdependent network cannot be solved in polynomial time (Section 3.4 - Theorem 1).

- A Deep reinforcement learning-based algorithm for Progressive Recovery (DeepPR) is proposed, incorporating the benefit of a baseline heuristic algorithm, RATIO (Section 3.7).

- Our simulation results indicate that DeepPR can achieve near-optimal solutions under not only random failures but also intentional failures under which RATIO cannot perform well. (Section 3.8)

- Our results suggest that the integration of reinforcement learning and a heuristic algorithm that is specifically designed for an optimization problem provides a means to solve optimization problems more effectively than the simple use of reinforcement learning or heuristic algorithms alone (Section 3.9).

3.2 Related Works

Many works have addressed the problem of improving the reliability of network slices or Service Function Chains (SFCs) through the proper placement and replication of functions [17, 55, 18]. However, interdependent networks could encounter larger failure events because of the cascading failures that a failure of a small portion of a network spreads throughout the entire network along with chains of interdependencies.

Pioneering work [76] on the progressive recovery problem focuses on determining the recovery order of communication links that maximizes the total flow on the recovered network with limited resources. As an extension, the work [53] proposes node evaluation indices to decide the recovery order to maximize the number of virtual networks accommodated. Considering the necessity of monitoring to observe failure situations, the joint problem of progressive recovery and monitor placement is discussed in [13]. The work [19] discusses an advanced recovery problem to maximize the overall reachability to content when considering the one-directional dependency between content and physical network equipment.

The fragility induced by dependency between networks consisting of different types of nodes has been studied in the context of interdependent network research [57, 39, 61]. In particular, the interdependency between virtualized nodes and physical nodes in optical networks is considered in [57]. A similar dependency caused by VNF orchestration is discussed in [39].

The works in [63, 21, 6] analyze the behaviors of failure propagations in such interdependent networks when each node performs local recovery (healing), where a functioning node substitutes for the failed node by establishing new connections with its neighbors.

Progressive recovery problems in interdependent networks have been discussed in [42, 83, 41, 37]. Classifying the progressive recovery problems by the types of interdependency, the work [42] proposes the optimum algorithm for a special case and heuristic algorithms for other cases. ILP and Dynamic Programming-based algorithms are employed to solve a variant of the progressive recovery problem in [83].

Other works [58, 48] propose some metrics to evaluate network nodes that can be used to decide the priority among the nodes.

3.3 Model

3.3.1 Network Model

A network, which consists of virtualized functions and infrastructure nodes hosting the functions, is modeled by an interdependent network that is formed by two constituent graphs $G_i = (V_i, E_{ii})$ $(i \in \{0, 1\})$, which correspond to the function layer (G_0) and the infrastructure layer (G_1). Multiple orchestration function nodes in G_0 are assumed to be replicas at geographically isolated servers for the purpose of providing protection against failures. We do not assume any other protection methods in the network. A pair of nodes in different constituent graphs can be connected by an arc representing their dependency relationships: A_{ij} $(i, j \in \{0, 1\},\ i \neq j)$. Edges in $E_{ii} \subseteq V_i \times V_i$ are called *intra*-edges because they connect pairs of nodes in a constituent network. In contrast, arcs in $A_{ij} \subseteq V_i \times V_j$ $(i \neq j)$ are called *inter-* or *dependency* arcs. An arc $(v_i, v_j) \in A_{ij}$ $(v_i \in V_i, v_j \in V_j)$ indicates that a node v_j has dependency on a node v_i. The node v_i is called a *supporting* node, and v_j is a supported node.

Two node attribute functions are defined to capture the characteristics of each node: *demand* and *utility* functions. The demand function $d : V \to \mathbb{N}$ represents how many resources need to be assigned to fully recover a given node. This demand can be interpreted as the cost or manpower to repair a specific node in the context of recovery problems. The utility function $u : V \to \mathbb{N}$ indicates the importance of a given node, such as the number of slices using the functions on it when it is fully recovered.

3.3.2 Network Failure and Progressive Recovery Plan

When a network failure event occurrs at time t_0, some nodes in the network become nonfunctional. Let $F[t_k] \subseteq V(= \bigcup_{i\in\{0,1\}} V_i)$ denote a set of nonfunctional nodes at time t_k. With this notation, the nonfunctional nodes right after the failure are represented as $F[t_0]$. A failure is represented by a node set in this paper because any failure of an edge can be converted to a node failure by replacing the nonfunctional edge $(v_i, v_j) \in E$ with a nonfunctional node v_{ij} and two functional edges $\{(v_i, v_{ij}), (v_{ij}, v_j)\}$.

In progressive recovery scenarios, we receive a limited amount of resources at each time step after a failure. The time steps can be understood as discretized time units staring at the time of the occurrence of an initial failure. The resource function $r : t_i \mapsto c_i \in \mathbb{N}$ indicates the amount of the repair resources available at time t_i ($i \in \{0, ..., T\} \subset \mathbb{N}$).

A progressive recovery plan P is an assignment of the available resources to the nonfunctional nodes. Formally, P is a $(T + 1) \times |V|$ matrix whose entries indicate the amount of resources assigned to a specific node at a specific time. Because of the limitation on the available resource amount, $P[t_i] \left(:= \sum_{v\in V} P[t_i][v]\right) = r(t_i)$ for every t_i.

During the recovery process, nodes can be classified by two measures: the amount of resources assigned to the node and the functionality of the node. A node v is *saturated* when it has received enough recovery resources: $d(v) \leq \sum_{i=0}^{k} P[t_i][v]$. Let $K[t_i]$ denote a set of saturated nodes at time t_i. A node v is said to be *functional* if and only if it is (1) saturated

and (2) reachable from at least one saturated supporting node in the other constituent graph via a simple path consisting of functional nodes. When a node v is functional at time t_i ($i \in \{0, ..., T\} \subset \mathbb{N}$) under a recovery plan P, the node state function $\alpha_i^P(v) = 1$; otherwise 0. A node v is *recovered* at t_i only when it becomes functional by assigning $P[t_i][v]$. In real networks, a nonfunctional saturated node can be interpreted as either an infrastructure node unreachable from an orchestration function or a virtualized function that is hosted on an infrastructure node that is nonfunctional. The total number of time steps, T, is always computable from the demand function d and the resource function r: $T = \sum_{i=1}^{\infty} 1_{(\sum_{v \in V} d(v) - \sum_{j=1}^{i} r(t_j)) \geq 0}$, where, for a given condition A, $1_A = 1$ if the condition A is satisfied; otherwise 0. This is because every node must be recovered when the demands of all nodes are satisfied.

A resource assignment $P[t_i]$ at each step t_i is called a *splitting* assignment when it prevents any nodes from saturation or recovery even though there exists a node that can be saturated or recovered at t_i. Contrarily, a *concentrating* assignment saturates or recovers some node if possible, and provides all the additional resources, which cannot saturate nor recover any node, to one unsaturated node.

3.4 Problem Formulation

This section formulates the progressive recovery problem in interdependent networks and discusses and proves some properties of the problem.

3.4.1 The Problem and Special Cases

The progressive recovery problem is to find a recovery plan P represented by a (time step $\times$ node)-matrix that maximizes the sum of utility provided by functional nodes during the recovery. Formally, the recovery problem is formulated as a combinatorial optimization

problem with a variable matrix P that determines the state $\alpha_i^P(v)$ of each node v at each time step t_i.

Problem 2. Progressive Recovery Problem (PR): Given a graph $N = (V = V_0 \cup V_1, A = A_{01} \cup A_{10} \cup E_{00} \cup E_{11})$, a demand function d, an utility function u, a set of initially failed nodes $F[t_0] \subseteq V$, and a resource function r, maximize the network-wide utility $U_P = \sum_{i=0}^{T} \sum_{v \in V} u(v)\alpha_i^P(v)$ by deciding a resource assignment matrix P.

A simpler case of the problem is one in which it is assumed that the functionality of virtualized functions totally depends on the functionality of a physical server hosting the function. In other words, there is no need for the assignment of recovery resources to repair virtualized functions since the unavailability of the functions occurs only due to the loss of physical servers hosting them. In our terminology when virtualized function nodes are nonfunctional, they are always saturated. The interdependency between the virtual and infrastructure network still exists even with the above assumption since any physical machine needs at least an indirect connection with a virtual control function. Obviously, a virtual function needs at least one physical machine, which can host it, to be functional. This scenario is formally defined as follows.

Definition 3. A graph $N = (V, A)$ in the progressive recovery problem is said to be *interdependency-embedded* when nodes in $G_0 = (V_0, E_{00})$ never require repair resources to be functional. In other words, nodes in G_0 are nonfunctional only because the loss of supporting nodes in the other constituent graph: $v \in K[t_0]$ for any node $v \in (V_0 \cap F[t_0])$.

3.4.2 Intractability

To prove the intractability of the problem, we discuss the hardness of solving a minimal case of the progressive recovery problem named StarPR.

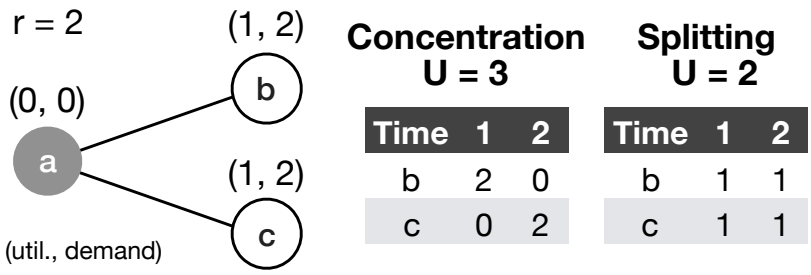

Figure 3.3. Concentration vs. Splitting in an interdependency-embedded star graph: Available resources at a time step should be concentrated to a set of nodes as much as possible.

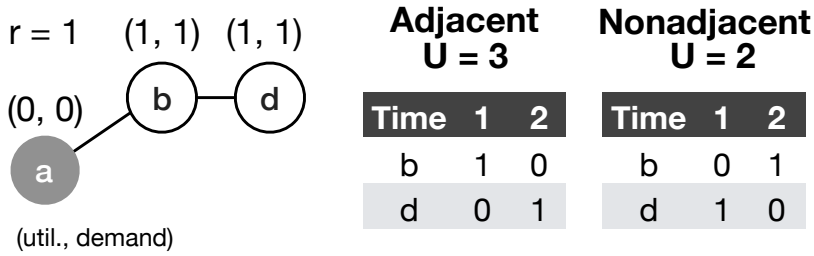

Figure 3.4. Allocation and the Adjacency to Functional Nodes in an interdependency-embedded tree graph: The nodes closer to functional nodes should be prioritized in a recovery order.

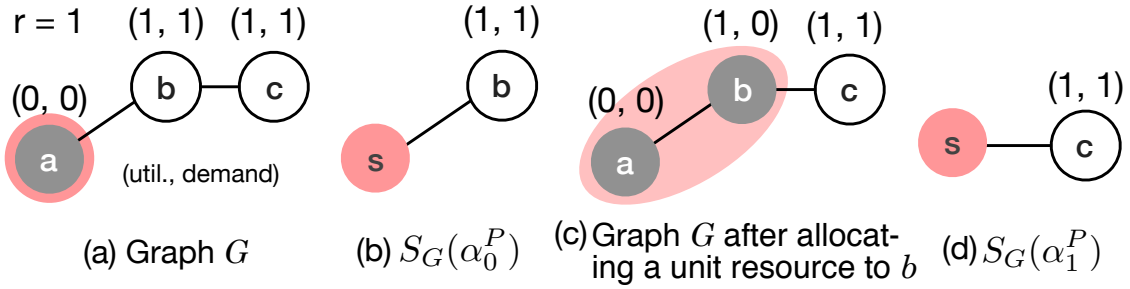

Figure 3.5. Pseudo Star Graphs.

Problem 3. Interdependency-embedded star case (StarPR): Assume that the graph topology is a star whose nodes are in G_1, except for the center node $v \in V_0$; also, each node $u \in V_1$ is biconnected with $v \in V_0$. An example is illustrated in Figure 3.3.

Definition 4. Time-Invariant Incremental Knapsack Problem (IIK) [8]: Let $X = \{x_i\}$ denote a set of items, which each have value $a(x_i)$ and weight $w(x_i)$. For any subset X' of X, the value and weight are defined as follows: $a(S) = \sum_{x_i \in S} a(x_i)$, and $w(X') =$

$\sum_{x_i \in S} w(x_i)$. IIK is to find a sequence of subsets of X, $[S_1, S_2, ..., S_T]$ ($S_i \subseteq S_{i+1}$, $i = 1, ..., T-1$) from time 1 to T that maximize $\sum_{t=1}^{T} a(S_t)$ subject to $w(S_t) \leq B_t \, (t = 1, ..., T)$, where B_t is the available capacity of the knapsack at time t. Note that IIK is known to be NP-hard.

Theorem 1. The interdependency-embedded star case (StarPR) is NP-hard.

Proof. We show IIK $\leq_p$ StarPR.

Given an instance of IIK, an instance of StarPR is constructed as follows. We construct a graph with v_i's that corresponds to each item $x_i \in X$ and a special node v. Edges are added so that each v_i is adjacent to v: $E = \{(v, v_i)\}$. Formally, $N = (\{v\} \cup \{v_i\}, E)$. The set of failed nodes F consists of v_i's. The demand d and utility u functions are defined using the given weight w and value a functions, respectively. The available resource function value $r(t)$ for time t is defined by the given capacity function B_t. This conversion is obviously executed in polynomial time.

Clearly, IIK reaches the optimum if and only if StarPR reaches the optimum since the objective functions of these two problems are identical with the settings above. The progressive property of StarPR, which accumulates utility over time, is inherited in the property of IIK solutions that $S_i \subseteq S_{i+1}$ $(i = 1, ..., T-1)$. □

Therefore, the PR problem is, in general, an NP-hard problem. This proof also implies that the intractability of a progressive recovery problem changes, depending on the d, u, and r functions. The work [42] provides a polynomial time optimum algorithm for the interdependency-embedded star case (Case 1 in [42]) with $r : t_i \mapsto C$ and $d : V \to C$, where C is a constant.

3.5 Problem Reduction with Interdependency Embedding

An instance of the progressive recovery problem is characterized by network topology, the distribution of damages (demand) and functional importance (utility), and the amount of

available resource. Moreover, the network topology could demonstrate multiple variations in the number of virtualized orchestration functions V_0, the connectivity between the function layer and the infrastructure layer, and the connectivity within the infrastructure network G_1.

The multitude of possible configurations of problem instances makes it difficult to provide a general algorithm to solve the problem with high performance. Therefore, we first limit the problem space to interdependency-embedded graphs, where only one orchestration function exists (Definition 3), and show that only a certain set of resource allocation actions can achieve the optimal solutions in such special graphs (Theorem 3). Next, it is shown that all the problem instances in which the virtual function nodes have *zero demand* can be reduced to the progressive recovery problem in interdependency-embedded graphs. Since the assumption that virtual functions do not suffer from the physical damage (i.e. the demand is 0) is reasonable, we can claim that it is enough to provide a general algorithm for the interdependency-embedded cases to solve the entire problem space of the progressive recovery problem in interdependent networks (Theorem 4).

An important fact about the interdependency embedding is that a resulting interdependency-embedded graph maintains the complex nature of interdependency between two layers in a simpler form. When deciding a recovery plan, the factor incurred by the interdependency is a requirement of the reachability from each infrastructure node to a virtualized orchestration function. An interdependency-embedded graph represents the requirement as a path from special nodes (V_0) to the other nodes. Hence, the heterogeneity of constituent nodes in interdependency-embedded graphs preserves the features of the recovery problem in interdependent networks.

Table 3.2. Summary of Theoretical Results: Theorems collaboratively claims that (1) the special case (the PR in interdependency-embedded graphs) captures the PR in general, and (2) a set of meaningful actions is characterized by resource concentration and adjacency between nonfunctional and functional nodes.

Question	**Lemma/Theorem**	**Key Assumptions**	**Reasoning**
Should we assign the resources at a step to one node or multiple nodes by splitting them?	Lemma 6: Concentration to one node is better.	Interdependency-embedded star graphs.	Earlier recovery starts incrementing the utility earlier.
Is it always better to assign the resources at a step to the nodes adjacent to the currently functional nodes?	Lemma 7: Assignments to adjacent nodes are better.	Interdependency-embedded tree graphs.	Saturation does not contribute to the utility.
Does the previous two statements hold for more general cases?	Theorem 2 Theorem 3	Interdependency-embedded tree graphs. Interdependency-embedded graphs.	Lemma 6, 7 Theorem 2, Definition 5
Can we convert the PR in general into the PR in an interdependency-embedded graph?	Theorem 4	$u(v) = 0 \ \forall v \in V_0$.	Lemma 8-10: The order of recovery is not affected even when converting a graph to an interdependency-embedded graph. (See Figure 3.7.)
How difficult is the PR problem?	Theorem 1 The PR is NP-hard.	Interdependency-embedded star graphs. Any graphs satisfying $u(v) = 0 \ \forall v \in V_0$.	Reduction from IIK (Definition 4). Theorem 4

Problem 4. Interdependency-embedded rooted tree case: Extend a star in Problem 3 by adding more nodes that are not adjacent to the node in G_0. i.e., the graph is a tree rooted at the node v in G_0. An example is illustrated in Figure 3.4.

Lemma 6. The optimum recovery plan P^* for any interdependency-embedded star graph only consists of concentrating assignments when $r : t_i \mapsto C$ $(\forall t_i)$. (See Figure 3.3.)

Proof. First, we argue that the statement is true for a star graph with two nodes where C is set to 2, and the demands of the nodes are divisible by C. Suppose P only consists of concentrating assignments, and P' includes some splitting assignments.

Because P concentrates resources on a node v_i, the node becomes functional after $\frac{d(v_i)}{C(=2)}$ steps. After these steps, it takes $\frac{d(v_j)}{2}$ additional steps to recover the other node v_j. Note that during these $\frac{d(v_j)}{2}$ steps, the network-wide utility is always $u(v_i)$. Therefore, $U_P = \frac{d(v_j)}{2} \times u(v_i) + u(v_j)$.

Consider P', which contains a splitting assignment at one time step t_k and concentrating assignments for the other steps. The splitting must be conducted before v_i becomes functional since there are only two nodes. Then, it takes $\frac{d(v_i)}{2} + 1$ steps to recover v_i and $\frac{d(v_j)}{2} - 1$ steps for v_j. Note that v_j receives one unit of resource at both step t_k and step $(\frac{d(v_i)}{2} + 1)$. Therefore, $U_{P'} = (\frac{d(v_j)}{2} - 1) \times u(v_i) + u(v_j) < U_P$. The same discussion can be applied to the cases with more splitting. Thus, $U_{P'}$ decreases when more splitting assignments are included in P'.

Second, we relax the settings by allowing more general demands: $d(v_i), d(v_j) \in \mathbb{N}$. Without loss of generality, suppose $d(v_i) > d(v_j)$. There are three recovery plans to be compared. Let P_l denote the recovery plan only consisting of concentrations with the prioritization of v_l and P' be a plan including splitting. Based on the previous discussion, $U_{P_i} = \left\lceil \frac{d(v_j)}{2} \right\rceil \times u(v_i) + u(v_j)$, and $U_{P_j} = \left\lceil \frac{d(v_i)}{2} \right\rceil \times u(v_j) + u(v_i)$.

When P' uses the splitting assignment at one step, v_j is recovered at step $\left\lceil \frac{d(v_j)}{2} \right\rceil$, and it takes $\left\lceil \frac{d(v_i)-1-x}{2} \right\rceil$ additional steps to recover v_i, where $x = d(v_j) \mod 2$. This is because

the splitting assigns one unit of resources to v_i, and the ceiling function at step $\left\lceil \frac{d(v_j)}{2} \right\rceil$ may assign another excess unit, depending on if $d(v_j)$ is divided by C. Therefore, U_P is at most $\left\lceil \frac{d(v_i)-1}{2} \right\rceil \times u(v_j) + u(v_i) \leq U_{P_j}$. When P' exploits more splitting assignments, the network-wide utility decreases as observed in the previous setting.

It is easily shown by a similar discussion that, for any $C(> 2)$, a recovery plan that only includes concentrating assignments is better than plans including splitting assignments. This is because the difference in resource amounts is just a problem of scaling of C and d. Thus, the inherent property of the spitting and concentrating assignments does hold even with any different C.

It is also obvious that similar discussions hold for general star graphs with n nodes. The key property here is that the splitting delays recovery of a certain node by assigning resources to more nodes even though the number of steps required to recover all nodes is fixed: $\left\lceil \frac{\sum_{v \in V} d(v)}{C} \right\rceil$. □

Lemma 7. The optimum recovery plan P^* for any interdependency-embedded rooted tree never saturates any node that is not adjacent to a saturated node; i.e., the candidate nodes for resource assignments are always adjacent to a saturated node when $r : t_i \mapsto C$ $(\forall t_i)$.

Proof. For contradiction, consider the case where saturation gives us better network-wide utility. Suppose there are two adjacent nodes v_i, v_j in a rooted tree, such that v_i is adjacent to a saturated node, but v_j is not.

First, we consider the case only with concentrating assignments. After saturating v_j, it takes $\left\lceil \frac{d(v_i)}{C} \right\rceil$ steps to recover v_i. During these steps, the utility provided by v_j remains 0. In contrast when v_i is recovered before v_j, it takes $\left\lceil \frac{d(v_j)}{C} \right\rceil$ to recover v_j, and v_i will provide utility of $u(v_i)$ at each of these steps. This generates contradiction since the number of total steps in both scenarios stays the same.

Second, let us try to improve the total utility, by introducing the splitting assignments, from $\left\lceil \frac{d(v_j)}{C} \right\rceil \times u(v_i) + u(v_j)$. However, this is impossible based on the discussion in star

graphs. When exploiting the splitting at one step, the duration that v_i is functional is strictly less than $\left\lceil \frac{d(v_j)}{C} \right\rceil$. □

Theorem 2. The optimum recovery plan P^* for any interdependency-embedded rooted tree only consists of concentrating assignments that allocate resources to nodes adjacent to a saturated node when $r : t_i \mapsto C$ $(\forall t_i)$. (See Figure 3.4.)

Proof. When a network has only one initially saturated node, Lemma 7 eliminates the possibilities to assign resources beyond the neighbors of the saturated node. Then, the network can be considered as a star graph consisting of the saturated node and its neighbors. Hence, the statement holds because of Lemma 6.

Accordingly, the node that becomes saturated next is adjacent to a saturated (functional) node. By contracting the edge between the two saturated nodes, the problem is reduced to the original problem with one saturated node. □

Definition 5. Pseudo star graph $S_G(\alpha_k^P)$: Given a graph $G = (V, E)$ and a node state function α_k^P at time t_k under a recovery plan P, the logical star graph $S_G(\alpha_k^P) = (V(S_G(\alpha_k^P)), E(S_G(\alpha_k^P))$ consists of one logical functional node s and the nodes adjacent to any of the functional nodes in original graph, and edges connecting s and the others. (See Figure 3.5.)

Using the pseudo star graph, we prove Theorem 2 can be extended to any interdependency-embedded graphs.

Theorem 3. For any interdependency-embedded graph, the optimum recovery plan P^* only consists of concentrating assignments that allocate resources to nodes adjacent to a saturated node when $r : t_i \mapsto C(\forall t_i)$.

Proof. It is trivial that the optimum recovery plan does not saturate any node that is not adjacent to a saturated (functional) node even when a graph has more than one initially

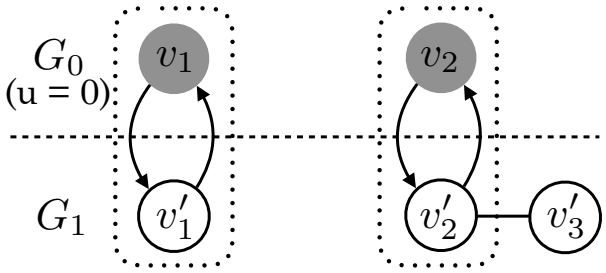

Figure 3.6. Supporting Pairs (v_1, v'_1) and (v_2, v'_2): The first recovery occurs only when two nodes in a pair are saturated.

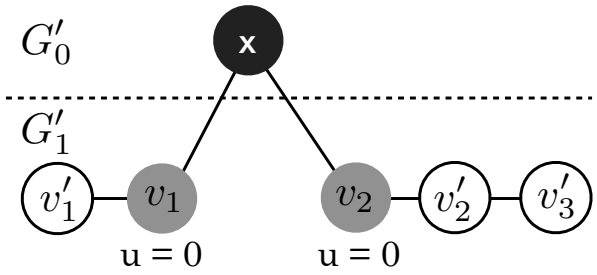

Figure 3.7. Conversion into an Interdependency-embedded Graph: The graph in Figure 3.6 is converted by adding a new node x, which forms the new function layer G'_0. The other nodes form the new infrastructure layer G'_1.

saturated node. Based on a discussion similar to Lemma 7, the saturation of a node adjacent to a functional node always provides more network-wide utility over time since the node to which resources are assigned starts contributing to the utility in an earlier step. Thus, the candidate nodes for resource assignment at each step t_k are the nodes adjacent to any saturated (functional) node.

Therefore, a resource assignment decision at each time step, $P[t_k]$ is equivalent to the progressive recovery problem in a logical star graph $S_G(\alpha_k^P)$, where α_k^P is a node state function reflecting recovery from t_0 to t_{k-1} in P. Therefore, it can be considered as the recovery problem in a star graph with a single logical functional node at the center (See node s in Figure 3.5.) and surrounding leaf nodes $V(S_G(\alpha_k^P))$.

Hence, it is easily provable, by the argument in Lemma 6, that the optimum plan does not involve splitting assignments since the concentration of the split resources to a node can always recover the node in an earlier time step and provide more network-wide utility. □

Next, we claim that the progressive recovery problem with any network topology can be converted into the progressive recovery problem in an interdependency-embedded graph.

Definition 6. A pair of nodes $v \in V_0$ and $v' \in V_1$ is called a *support pair* when $(v, v') \in A_{01}$ and $(v', v) \in A_{10}$. (See Figure 3.6.)

Lemma 8. When v and v' are the first support pair recovered in a given graph N, the order of saturation of these two nodes does not influence the total utility.

Proof. Let us assume that a recovery plan saturates v first and v' later. Note that there may be some nodes saturated before and between v and v'. Since v and v' are the first supporting pair to be recovered, there is no functional node in N before v' is saturated. The total utility generated until the step t_i when v' is saturated is $u(v)+u(v')+\sum_{w \in V_r} u(w)$, where $V_r \subseteq K[t_i]$ is a set of saturated nodes that are reachable from v or v'. When we exchange the ordering of v and v', the total utility until the step t_i when v is saturated remains the same because the saturated nodes by t_i are same. Therefore, the order of saturation of v and v' does not change the total utility. □

Lemma 9. In any graph, the first two nodes saturated by the optimum recovery plan P^* are always the nodes in a support pair.

Proof. For contradiction, assume a node w was a node saturated at first by the optimum recovery plan P^*, and the two nodes v, v' in a support pair will be recovered right after w. Without loss of generality, it is assumed that v is saturated first from Lemma 8. Then, the total utility until the step t_i when v' is saturated is $u(v) + u(v') + \beta u(w)$, where $\beta = 1$ iff w is adjacent to v or v'; otherwise, 0.

However, another recovery plan P', which saturates v and v' first and w later, provides the total utility until t_i of $s_w(u(v)+u(v'))+\beta u(w)$, where s_w is the number of steps required to saturate node w since v and v' are already functional at t_{i-s_w}. This contradicts the fact that P^* is the optimum. □

Lemma 10. In the interdependency-embedded rooted tree where any node adjacent to the node $u \in V_0$ has utility of zero: $d(v) = 0$ ($\forall v$ s.t. $(u, v) \in A_{01}$), the second node v_2 recovered by the optimum recovery plan P^* has utility strictly greater than zero: $d(v_2) > 0$.

Proof. All the nodes adjacent to $u \in V_0$ have the utility of zero. Therefore, the first node v_1 recovered by the P^* is one of these nodes. For contradiction, assume the second node v_2 is also one of these zero-utility nodes, and let v_k (k-th node recovered in the plan) be the first node recovered whose utility is greater than 0.

In order to recover v_k, it is necessary to have a zero-utility node that is already recovered for the reachability to u. There are two possible scenarios: (1) v_1 is adjacent to v_k, or (2) v_j $(2 \leq j < k)$ is adjacent to v_k.

For the first scenario, we can exchange the recovery order of v_2 and v_k. This exchange has no influence on the candidate nodes at each step after k-th recovery because the recovered nodes until k-th recovery stay the same. However, it increases the utility and contradicts the fact that P^* is optimum.

For the second scenario, we can exchange the recovery order of v_1 and v_j. Again, this does not change any candidate sets for recovery after k-th recovery. Since v_j is recovered at the very beginning, we can use the same discussion with the first scenario. Therefore, it provides a contradiction. Therefore, the second node recovered in the optimum plan should have a nonzero utility. □

Theorem 4. A progressive recovery problem with any general graph with $u(v \in V_0) = 0$ has an equivalent progressive recovery problem with an interdependency-embedded graph.

Proof. The problem with a general graph is converted into the problem with an interdependency-embedded graph as follows. We add a new node x to G'_0 and put all the nodes and edges in the original N into G'_1. An edge is added between $x \in V'_0$ and each $v \in \hat{V}'_1$, where $\hat{V}'_1$ consists of nodes that are originally in V_0 of N; i.e. $u(v \in \hat{V}'_1) = 0$. Figure 3.7 illustrates an example of constructing an interdependency-embedded graph from the graph shown in Figure 3.6.

Lemma 9 shows that the first two nodes to be saturated (recovered) are the ones in a support pair. Also, according to Lemma 8, it can be assumed without loss of generality that a node v in V_0 in each support pair is the first node to be saturated.

Algorithm 3 RATIO(G, F_{t-1})

Input: A graph $G = (V, E)$, A set of nonfunctional nodes F_{t-1}

1: $W = \text{neighbor}(V \setminus F_{t-1})$

2: Sort W based on $\dfrac{u(v_i)}{d(v_i)}$ in the decreasing order

3: Perform concentrating allocation from the head of W

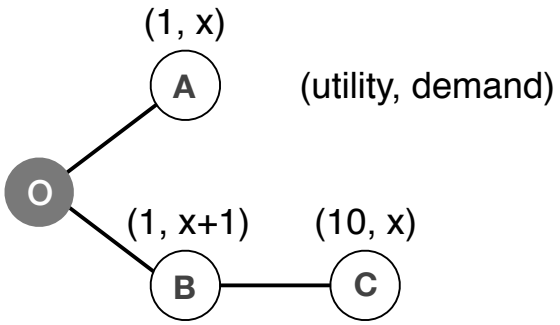

Figure 3.8. An Adversarial Toy Example: The worse utility-demand ratio of node B hides, from RATIO, node C that potentially produces higher overall utility even when compensating for the loss by selecting node B.

The newly added edges confirm that the first node recovered is one of the nodes in V_0 since x is the only saturated node in the initial step. The other correspondence between the two problems to be checked is that the second node recovered in N' is v' that forms a support pair with v in the original graph N, which is guaranteed by Lemma 10. □

Therefore, it is sufficient to consider cases of interdependency-embedded graphs. Also, it is possible to aggregate multiple nodes in G_0 into one logical node in G_0 to decide the resource assignment, as the proof of Theorem 3 suggests. Thus, without loss of generality, the rest of this paper only deals with the interdependency-embedded graphs with one node in G_0.

3.6 Baseline Heuristic for Progressive Recovery

This section describes a simple heuristic algorithm named RATIO that demonstrates promising performance to solve the progressive recovery problem under most random failures. It

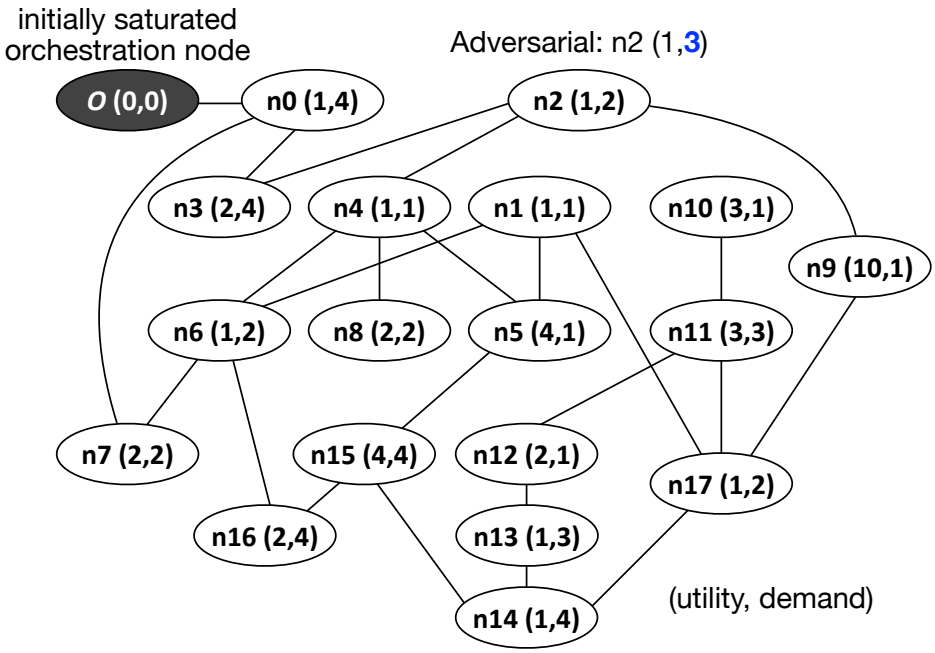

Figure 3.9. The IBM Network with Node Attributes for Preliminary Simulations: The node attributes inside of each node are used for the compliant scenario. In an adversarial scenario, node n2 is intentionally attacked to make its demand 3.

is also shown that the performance of RATIO can be arbitrarily degraded by certain failure settings that may be used in an adversarial way. Later, RATIO is incorporated as a base algorithm into a more advanced algorithm that can deal with such adversarial cases.

3.6.1 RATIO Heuristic and its Limitation

RATIO is a greedy heuristic algorithm inspired by the approximation algorithm of the set cover problem. This heuristic assigns resources to the most cost-effective nodes among the nodes adjacent to functional nodes at each time step by calculating the effectiveness, $\frac{u(v)}{d(v)}$. Algorithm 3 shows the pseudo code of RATIO.

To understand the behavior of RATIO, we consider two failure scenarios: (1) compliant failure scenarios into which most random failures fall, and (2) adversarial failure settings that could be made by a small change to the compliant cases. The adversarial setting can

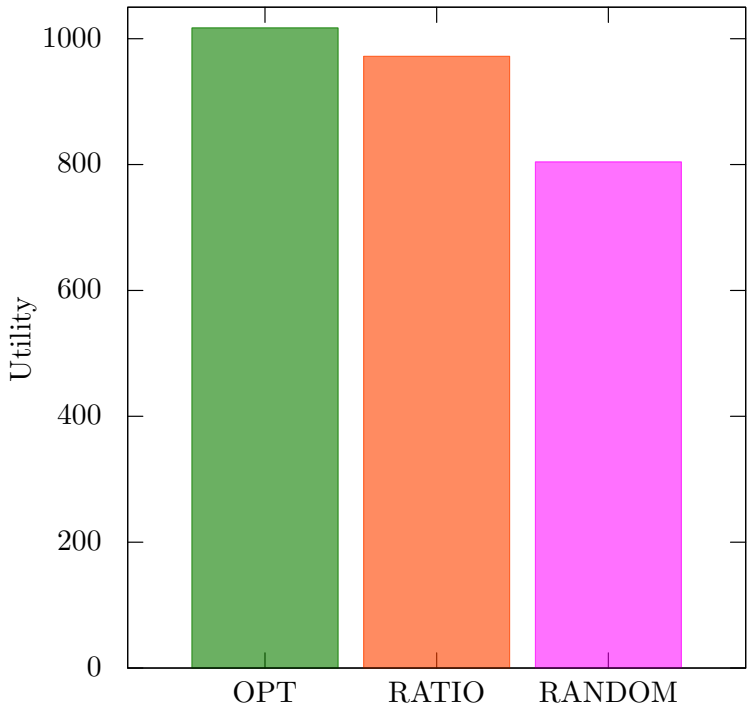

Figure 3.10. Total Utility under Compliant Settings (the IBM Network): RATIO could achieve the near-optimal solutions.

be defined as a structure where one node v_B, which is closer to currently functional nodes and with low effectiveness, hides other nodes with higher effectiveness beyond v_B.

Figure 3.8 illustrates a toy example of such adversarial scenarios. Suppose that one unit of resource is available at each time step ($r = 1$) and node O is a saturated function node. All the other nodes are initially nonfunctional, and their demands are depicted in the figure. Note that node B corresponds to the node v_B described above that hides some more effective node. At the first round of recovery, RATIO chooses node A between node A and B since $\frac{1}{x} > \frac{1}{x+1}$ ($x \geq 1$). It takes $\frac{x}{r}$ ($= x$) time steps to recover A. Then, RATIO recovers node B and C in order, which takes $x + 1$ and x steps, respectively. Therefore, the total utility of RATIO is always $u(A) \cdot (2x + 1 + 1) + u(B) \cdot (x + 1) + u(C) \cdot 1 = 3x + 13$. In contrast, the optimum strategy is recovering nodes B and C first, and then node A. In this case, the total

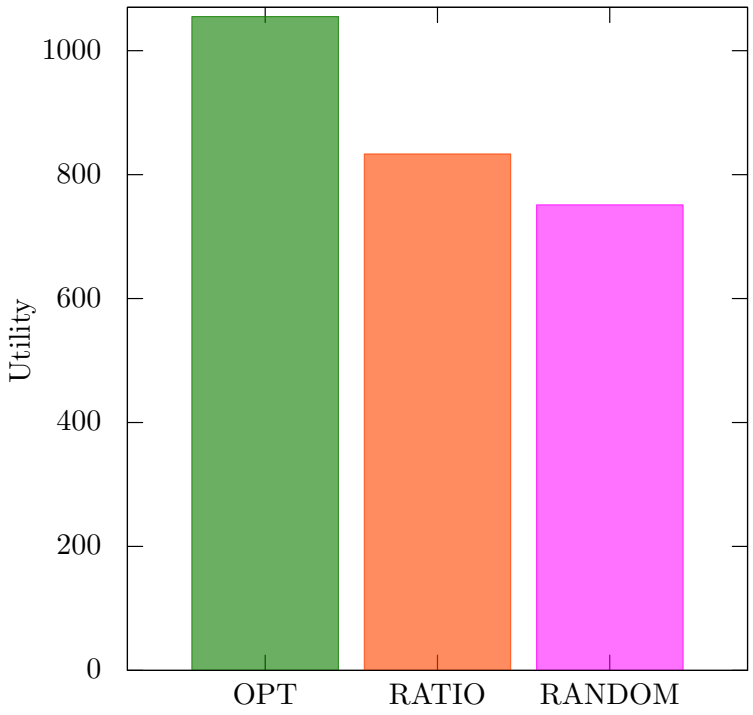

Figure 3.11. Total Utility under Adversarial Settings (the IBM Network): The performance of RATIO is easily deteriorated with small changes of the attributes of some nodes.

utility is $u(B) \cdot (2x+1) + u(C) \cdot (x+1) + u(A) \cdot 1 = 12x + 12$. When x becomes larger, the total utilities of RATIO and the optimum will diverge more drastically.

Revisiting the application scenario where a system consists of VNFs managed by MANO and infrastructure servers, the change in the demand of a node should be understood as the difference in the degree of physical damages that the node experienced. Since the minimal adversarial example is quite simple, a random failure incident in a larger network could contain it as an embedded substructure. Furthermore, it could be said that RATIO is vulnerable to failure events by malicious attacks since an attacker can easily embed this substructure in an attack and deteriorate the interim utility arbitrarily by setting x as large as possible. This scenario can be interpreted as an intentional attack where an attacker tries to hide a node with a high utility value v from RATIO by imposing more damages to the neighbors of v.

3.6.2 Preliminary Simulation Results

Figures 3.10 and 3.11 illustrate the total utility obtained by three methods in the IBM network [71] with compliant and adversarial settings shown in Figure 3.9, respectively. The adversarial setting is realized by a change in the demand of n2, as shown in the figure. The three methods are OPT, which is the theoretical optimum, RATIO, and RANDOM, which uniformly randomly chooses one node to allocate a unit of resource. For simplicity, the amount of resources at each time step is set to 1. RATIO only achieves 79.0% of the optimum with the adversarial setting although it approximately reaches 95.6% of the optimum with the compliant scenario. It is noteworthy that a slight change in the demand of one node (from 2 to 3) can worsen the performance of RATIO to this extent.

3.7 DeepPR: Reinforcement Learning for Progressive Recovery

We have focused on the following two properties of the progressive recovery problem to devise a robust algorithm, i.e. an algorithm that is robust against adversarial attacks.

- It is straightforward to compute the utility of a given recovery plan with a problem instance, which implies that we can generate a large data set consisting of recovery plans and the corresponding utility obtained by the plans.
- Observing the fact that such a simple baseline algorithm (RATIO) performs well in the case of random failures, it seems reasonable to complement RATIO with another algorithm for its adversarial scenarios.

Note that the first property is guaranteed by the NP-completeness of the decision version of the progressive recovery problem.

The first property suggests the use of a learning algorithm, which can fully receive benefits of generated examples of recovery plans. Most classical optimization methodologies for

static problems do not allow us to effectively utilize historical experiences. In particular, reinforcement learning seems one of the most suitable methods since our goal is to decide a particular action (recovery plan) given a current network state while considering the changes in the total utility depending on the recovery plan. Furthermore, most reinforcement learning algorithms have flexibility in how to explore a huge action space for learning. This flexibility enables us to incorporate the benefits of RATIO. The following summarizes key elements of our proposal, a Deep reinforcement learning algorithm for the Progressive Recovery problem (DeepPR).

3.7.1 Q-Learning

Reinforcement Learning (RL) is a method to learn the best mapping of states $\mathcal{S}$ to actions $\mathcal{A}$. The key elements of RL include the *agent*, who learns the mapping of state action pairs to numerical rewards for its trial actions, and the *environment*, which updates states and returns the numerical *reward* depending on actions the agent takes.

In Q-learning, the mapping is learned using the *action-value function* $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that represents the quality of each state-action pair. In theory, the Q-value of a state-action pair converges to $Q^*(s, a)$ after infinite trial actions (experiences): $Q^*(s, a) = \max_\pi \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi]$, which is the expected reward achievable by following the optimum action sequence (policy) π from state s taking action a at time t. Note that γ is a discount factor for future rewards that defines the learning horizon.

3.7.2 Deep Q-Network (DQN)

Mnih et al. [44] report a significant improvement in RL by introducing the Deep Q-Network (DQN). Instead of explicitly calculating Q-values, a DQN uses neural networks (NNs)—parametrized by a weight function θ—as a function approximator to estimate the optimum Q-values: $Q^*(s, a) \approx Q(s, a; \theta)$.

The dramatic improvement by DQNs in learning performance is achieved mainly by introducing *experience replay* and *Target-Net* [44]. The ϵ-greedy exploration method was also employed to effectively explore the state-action space.

Experience Replay

It is known that the correlation among experiences, which each are represented by quadruples $e_t = (s_t, a_t, r_t, s_{t+1})$ of a state, action, reward at time t, as well as a resulting state at time $t+1$, causes fluctuations in the learning process. Experience Replay buffers all the experiences $B = \{e_t\}$ and takes random samples from B for the Q-value updates. This random sampling prevents DQNs from undergoing fluctuation in training due to learning from correlated sequential experiences.

Target-Net and Eval-Net

In order to stabilize the learning, it is proposed to use two separate DQNs; one named Eval-Net for learning from each sampled experience, and the other, named Target-Net, for calculating the target Q-values. The weight function θ_T of Target-Net is periodically updated by copying the weight function θ of Eval-Net.

For each sampled experience e_t, the parameters of Eval-Net are updated by any gradient method with respect to the loss function $L(\theta)$, which represents the difference between the Q-values estimated by Eval-Net and Target-Net.

$$L(\theta) = \mathbb{E}_{e_t \sim U(B)}\left[\left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_T) - Q(s_t, a_t; \theta)\right)^2\right],$$

where $e_t \sim U(B)$ indicates the random sampling of $e_t = (s_t, a_t, r_t, s_{t+1})$ from the buffer B by the experience replay.

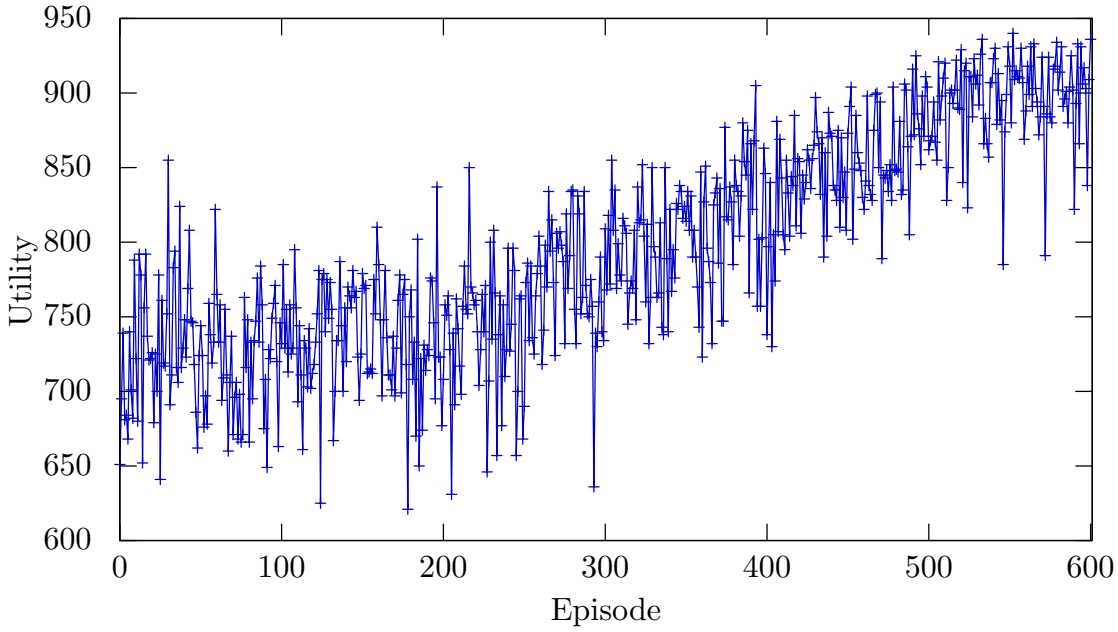

Figure 3.12. Learning Curve of DeepPR: A GNP random graph is given a compliant attribute setting.

ϵ-greedy Exploration

The tradeoff between exploration and exploitation is one of the crucial challenges in RL. The ϵ-greedy exploration is a commonly used approach to address this challenge. In this greedy approach, the agent follows the current best action known in a current state to reinforce the previous learning (exploitation) with probability $(1 - \epsilon)$. With probability ϵ, it tries an exploration by taking an action that is not determined by the previous learning.

DeepPR integrates two simple algorithms to realize the exploration; namely, RATIO and RANDOM. DeepPR chooses the best action following the RATIO heuristic with a predefined probability ω_{RATIO} and selects a random action from a legal action set with probability $(1 - \omega_{\mathrm{RATIO}})$. Therefore, the probability for DeepPR to take an action based on RATIO is $\epsilon\,\omega_{\mathrm{RATIO}}$.

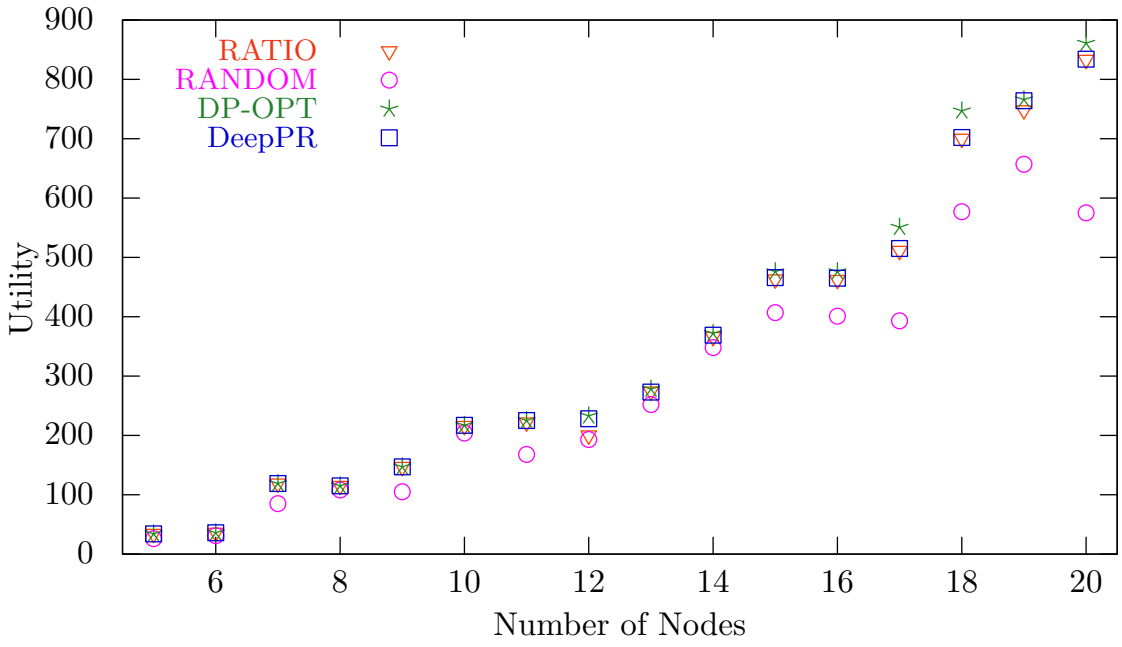

Figure 3.13. Utility under Compliant Settings (GNP Random Graphs): Both DeepPR and RATIO approach the theoretical optimum while DeepPR demonstrates slightly better results.

3.7.3 Applying DQN to PR

In our problem, the agent tries to learn the optimum resource allocations to nonfunctional nodes. Therefore, the legal actions for our agent are selecting a subset of nonfunctional nodes. Here, we assume a situation where at most one node is fully recovered at a time step by setting $d(v_i) + d(v_j) > 2C - 1$ $(\forall(v_i, v_{j\neq i}) \in V \times V)$, where C is the amount of available resources at a time step. Therefore, each **action** at time step t_k is represented as an ordered pair of nodes $[v_i, v_j]$, where the first node v_i is assigned $\min\{r(t_k), d(v_i)\}$ resources, and the second node v_j receives the remaining resources if they exist. Each **state** is represented as a $(|V|\times 1)$ vector in which the ith element indicates the remaining demand of the corresponding node $v_i \in V$. The **reward** of a state-action pair is the sum of utilities of the functional nodes.

A challenge in our problem is the size of the state-action space, which grows exponentially in the number of nodes. For example, a graph consisting of 20 nodes with a minimal demand setting, where $d(v) = 1$ $(\forall v \in V_1)$, has over one million $(\approx 2^{20})$ possible states, assuming

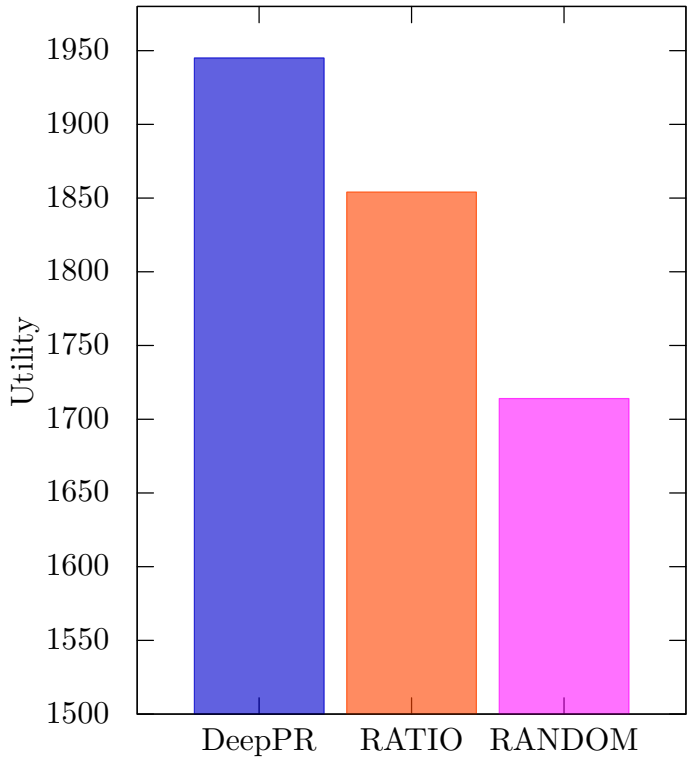

Figure 3.14. Utility under Compliant Settings (the BT North America Graph).

$r(t_i) = 1$ $(\forall i \in [0, T])$. This is minimal since the size of the state space increases with the range of $d(v)$. The number of legal actions is at most the number of 2-permutations of V, $P(|V|, 2)$. Therefore, the number of state-action pairs is approximately $2^{20} \times P(20, 2)$. In order to improve the performance of exploration, the integrated exploration, which comprises RANDOM and RATIO, is adopted in DeepPR with appropriate ϵ and ω_{RATIO}.

Our theoretical results help us to reduce the number of actions to be considered at each time step in practice since nodes that are not adjacent to currently functional nodes are automatically excluded from the candidate nodes to allocate resources. Therefore, the size of a set of nodes W_i that are potentially allocated resources at each time step t_i is $|W_i| = \sum_{v \in V \setminus F[t_i]} |\text{neighbor}(v)|$, where neighbor$(v)$ is the set of nodes adjacent to v.

3.8 Evaluations

Simulations are conducted with different topologies and node attributes, as explained below. DeepPR is also evaluated under both compliant and adversarial failures, and is compared with the theoretical optimum (DP-OPT), RATIO, and RANDOM.

3.8.1 Simulation Settings

Network Topology

GNP random graphs [47], the BT North America graph [70], and the IBM graph [71] are used as network topologies. Since our theoretical results indicate it is enough to test the algorithm performance in interdependency-embedded graphs with a single node in G_0, a node in G_0 is randomly selected in each graph. For GNP random graphs, the following ranges are used: $p = 0.2$, and $n \in \{5, 6, ..., 20\}$ for compliant scenarios and $n \in \{5, 6, ..., 34\}$ for adversarial settings. Note that only connected GNP random graphs are fed into our simulations. The BT North America graph is based on an IP backbone network with 36 nodes and 76 edges. The IBM graph is a backbone network consisting of 18 nodes and 16 edges.

Node Attributes and Available Resource

The utility, demand, and resource values are randomly selected among the integers within given ranges for GNP random graphs and the BT North America graph. For GNP graphs, we have conducted simulations with the following settings: (utility range, demand range, resource amount available at each time step) = $([1, 4], [1, 2], 1)$, $([1, 10], [2, 4], 3)$, $([1, 10], [2, 4], 1)$, $([1, 5], [1, 5], 3)$, $([1, 5], [1, 5], 1)$. As similar trends are observed in all the settings, this paper only describes the results under $([1, 4], [1, 2], 1)$. The setting for the IBM graph follows the node attributes shown in Figure 3.9. Note that all the nodes in both graphs are assumed to be initially nonfunctional: $F[t_0] = V$.

DQN Settings

Our DQN consists of three fully connected layers: the input, middle, and output layers with $|V|$, 200, and $P(|V|, 2)$ neurons, respectively. The input layer receives the state vector, which represents the remaining demands, and the output layer indicates the evaluation of possible legal actions for a given state. The Rectified Linear Unit (ReLU) is used for the activation function in both middle and output layers in the DQN, and the reward discount factor γ is set to 0.6. The training of the DQN is conducted by the Adam algorithm (AdamOptimizer in TensorFlow [69]) that minimizes the estimation loss $L(\theta)$.

For exploration, ϵ is initially set to 1.0, decreasing by 0.0001 after every episode until 0.1. This encourages DeepPR to visit more diverse state-action pairs at the beginning and to reinforce its learning as it has experienced more. Additionally, ω_{RATIO} is fixed to 0.5 to incorporate RATIO in exploration.

The hyperparameters and settings are selected based on the cross validation performance. In particular, we conducted experiments to empirically determine the best combination of activation functions in the middle and output layer, considering functions typically used for classification problems: ReLU, Leaky ReLU, tanh, and softmax. It is observed that all combinations demonstrated similar performance, with the exception that the use of tanh or softmax in the output layer has a slightly negative impact on performance.

3.8.2 Simulation Results

Figure 3.12 illustrates a sample of the learning curve of DeepPR over episodes, which are alternating sequences of states and actions from the initial network state to the fully recovered state. This sample is obtained in a GNP graph with 19 nodes, and similar curves are also observed in other graphs. Since the NNs are randomly initialized, the initial Q-values do not reflect the actual rewards. Through the update on Q-values and explorations, the NNs are trained to select an action that maximizes the total utility. In the figure, the utility (total

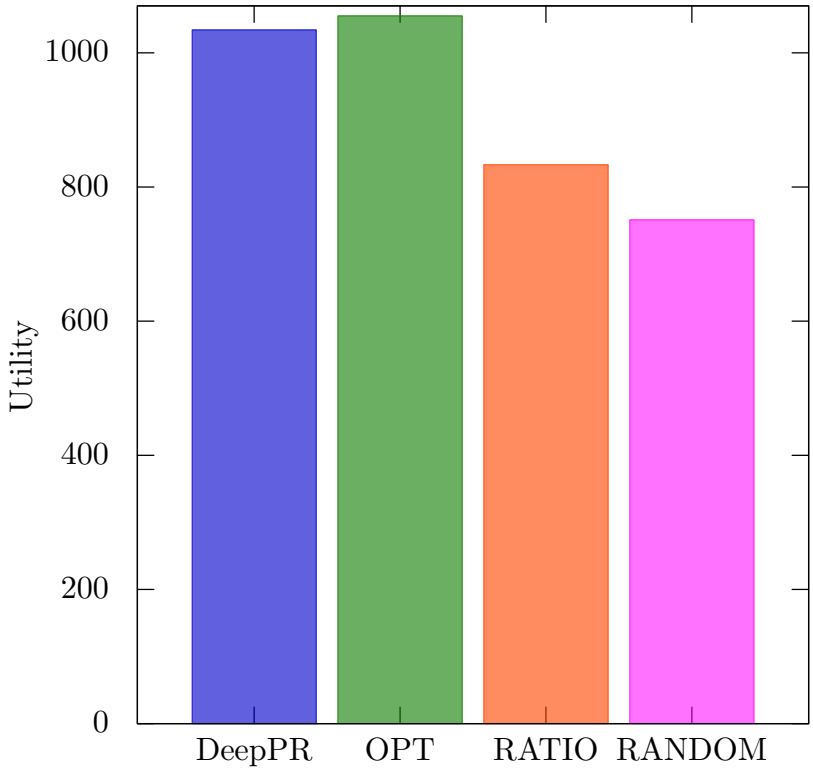

Figure 3.15. Utility under Adversarial Settings (the IBM Graph): DeepPR shows the performance similar to what it demonstrated with compliant scenarios although RATIO suffers from the adversarial attack.

reward) that DeepPR achieves stays at approximately 725 until around the 250th episode, and after that, it continues increasing towards around 900. Because of the exploration by random actions, utility values fluctuate during the entire training period. Note that each episode takes 1.057 seconds on average on a computer with a 2.5 GHz Intel Core i5 CPU, Intel HD Graphics 4000 (1536 MB), and 8 GB memory.

Figure 3.13 indicates a comparison among the four algorithms in terms of total utility in GNP random graphs. In smaller graphs, the utility obtained by DeepPR always matches with the theoretical optimum (DP-OPT). In theory, Q-learning is guaranteed to achieve the optimum by visiting each state-action pair an infinite number of times. Since it is easier to visit each state-action pair a greater number of times in graphs with fewer states and action choices, the estimation of Q-values seems to converge to more accurate values, which leads

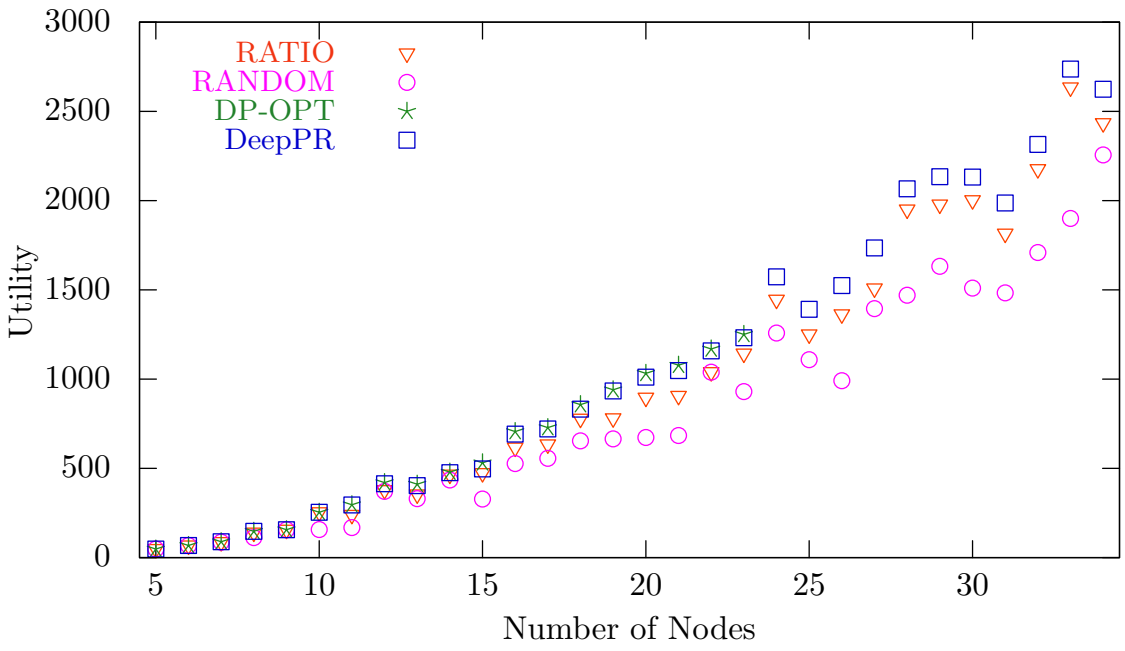

Figure 3.16. Utility under Adversarial Settings (GNP Random Graphs): DeepPR obtains the solutions closer to the optimum while the difference between DeepPR and RATIO increases from the compliant cases.

to the optimum. In contrast, the difference between DP-OPT and DeepPR increases in some larger graphs for the same reason. Compared to RATIO, DeepPR performs slightly better in those larger graphs. DeepPR achieves 96.6% of the optimum on average in the graphs of size 15 to 20 nodes while RATIO reaches 95.8% of the optimum in the same graphs. Also, RANDOM is the worst heuristic among the four methods over all sizes of graphs and continues getting worse along with the graph size because of the increase in legal actions.

Figure 3.14 shows the utility obtained by three algorithms in the BT North America graph. Here, DP-OPT is not included since it is intractable due to the number of nodes. In this practical topology, we also observed a trend similar to the results from GNP graphs. As mentioned in Section 3.6, all simulation results indicate that RATIO could attain the utility values close to the utility obtained by DeepPR and the optimum when input node attributes do not contain an adversarial substructure.

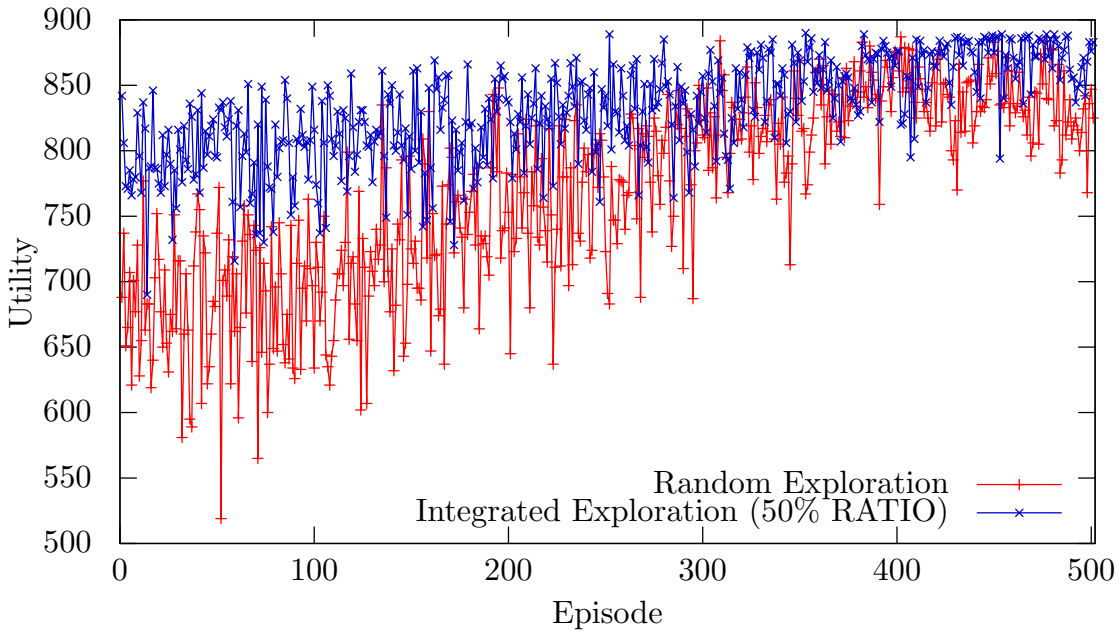

Figure 3.17. Learning Curves with Different Exploration Methods (A GNP Random Graph): The utilization of RATIO as one of the exploration methods ($\omega_{\text{RATIO}} = 0.5$) helps DeepPR to reach a better solution in earlier episodes.

In contrast, the total utility obtained by RATIO is easily deteriorated by intentional attacks with the adversarial settings as shown in Figure 3.15 and 3.16. However, DeepPR demonstrates robustness in these scenarios; i.e., DeepPR continues to achieve solutions closer to the optimum as it does in the previous compliant cases. For example, the total utility obtained by DeepPR in the GNP graph with 23 nodes is 98.5% of the optimum though RATIO obtains 91.4% of the optimum. This is because DeepPR still explores other possible state-action pairs by random exploration with probability $\epsilon(1 - \omega_{\text{RATIO}})$. The difference in performance between DeepPR and RATIO diverges in GNP random graphs with the adversarial settings, as shown in the figure.

3.9 Discussions

The total utility achieved by DeepPR surpasses that of the other methods since DeepPR integrates random actions with a well-behaving heuristic, RATIO, to seek new experiences. This

randomness prevents the proposed method from degrading under the adversarial scenarios that cause RATIO to underperform.

Another benefit observed is the learning speed of Deep RL. In general, it is more difficult to train a NN as a Q-value estimator for discrete state-action spaces than to do so for continuous spaces due to the sparser relation among neighboring points. Therefore, more episodes could be required to obtain a more accurate estimator if we adopt a sample random exploration. However, our result indicates that it is possible to speed up the learning process by integrating a well-behaving heuristic that is specific to each problem. Figure 3.17 compares the utilities at each episode of DeepPR with the simple RANDOM exploration and the integrated exploration ($\omega_{\text{RATIO}} = 0.5$). Clearly, the exploration is conducted more effectively when a heuristic is incorporated, and higher utility values are achieved in earlier episodes.

There is no silver bullet to determine the most effective rate for the integrated exploration ω_{RATIO} due to the tradeoff between learning speed-up and exploration. We observed that the increase in ω_{RATIO} continuously reduces the number of episodes required by DeepPR to achieve a near-optimal solution. However, it also increased the probability for DeepPR to be stuck at a suboptimal solution when ω_{RATIO} approaches 1 because the action space during learning becomes limited to the actions defined by RATIO.

Our results suggest the applicability of Deep RL to a wide range of optimization problems for which some heuristic algorithms are proposed. When a heuristic—which performs well for compliant cases and suffers from some critical cases—is known for the problem, it seems obvious that the addition of some degree of randomness in the action selection helps the algorithm to receive important experiential feedback, especially from the critical cases. Deep RL, in general, can extract the characteristics of such feedbacks and remembers them for future actions. Therefore, our results imply that the integration of such a simple heuristic algorithm and the Deep RL technique would provide a general methodology to design

algorithms for such optimization problems, which may outperform the existing heuristic. Additionally, from the previous discussion, the integrated exploration can be more effective than general exploration methods due to problem-specific tuning.

3.10 Conclusion

This paper discusses the progressive recovery problem of interdependent networks to maximize the total available computation utility of the networks, where a limited amount of resources arrives in a time sequence. It is proved that the recovery problem with a general network topology always has an equivalent progressive recovery problem with an interdependency-embedded graph, which is much simpler but still NP-hard. Through a preliminary simulation result, it is also shown that a simple heuristic algorithm called RATIO, which determines the resource allocation based on the ratio of utility and demand, can perform well when the network does not contain a specific substructure. To cope with adversarial scenarios, we introduce DeepPR, a deep reinforcement learning-based algorithm whose exploration strategies are based on RATIO. The simulation results indicate that DeepPR achieves the near-optimal solutions in smaller real and random networks and is robust against the adversarial cases. Furthermore, it is empirically shown that the integration of RATIO and reinforcement learning improves the effectiveness of exploration of the learning. Our success in the integration suggests possible improvements of existing heuristic approaches for general optimization problems using reinforcement learning.

CHAPTER 4

ONLINE CONVEX OPTIMIZATION-BASED DYNAMIC BANDWIDTH ALLOCATION FOR PON SLICING WITH PROVABLE PERFORMANCE GUARANTEES

The emergence of diverse network applications demands more flexible and responsive resource allocation for networks. Network slicing is a key enabling technology that provides each network service with a tailored set of network resources to satisfy specific service requirements. The focus of this work is the network slicing of access networks realized by Passive Optical Networks (PONs). In this chapter, a learning-based Dynamic Bandwidth Allocation (DBA) algorithm is proposed for PON access networks, considering slice-awareness, demand-responsiveness, and allocation fairness. Our online convex optimization-based algorithm learns the implicit traffic trend over time and determines the most robust window allocation that reduces the average latency. Our simulation results indicate that the proposed algorithm reduces the average latency by prioritizing delay-sensitive and heavily-loaded ONUs while guaranteeing fairness of the resource allocation.

4.1 Introduction

Emerging 5G networks are being designed to support a diverse array of network applications. Enhanced Mobile Broadband (eMBB) targets up to 10 Gbit/s download speeds for mobile devices, while massive Machine Type Communications (mMTC) handles dense connections for IoT and M2M applications, and Ultra Reliable Low Latency Communications (URLLC) involves strict latency constraints [81]. An approach for accommodating massive and diverse connections in 5G access networks is to implement network slicing in Passive Optical Networks (PONs) [72]. Optical access networks have attracted many industrial attentions, as they are expected to reduce operational expenditure (OPEX) [3]. In particular, the current

development of C-RAN technology for mobile networks accelerates the benefit of such optical access networks [11, 54].

A PON consists of an Optical Line Terminal (OLT) and multiple Optical Network Units (ONUs) as depicted in Figure 4.1. The OLT and ONUs may be logically sliced so that one physical network can host multiple network infrastructure slices that are tailored for various types of applications. In this slicing scenario, additional consideration should be made to prioritize delay-critical network slices. For example, if a network slice hosts a URLLC network service, the ONUs on the slice should receive more bandwidth to meet the strict delay requirement on the order of 1 ms for both upstream and downstream traffic [46].

Time Division Multiplexing (TDM) for the upstream traffic from ONUs to an OLT is facilitated by a *Dynamic Bandwidth Allocation* (DBA) algorithm that adjusts time windows for each ONU to minimize the latency of upstream traffic [62]. Most existing DBA algorithms periodically solve a static optimization problem to minimize the traffic latency by choosing an optimal bandwidth allocation given traffic information of the ONUs [27, 23, 77, 26]. The traffic information can be an estimation of the future traffic amount at the time when the ONU starts sending traffic to the OLT. The estimation is computed based on historical data, the amount of remaining data in queues, and/or neural network-based traffic prediction. Nevertheless, errors in such estimations may lead to suboptimal solutions for the bandwidth allocation problem. The suboptimality ironically originates from the fact that such estimation-based algorithms try to achieve the optimum window allocation in each static optimization problem, which is defined based on the estimated traffic demand. Since the allocation is optimum only for the estimated demand, the discrepancy between the estimation and the real traffic patterns is directly reflected in the performance of the algorithms.

Therefore, we propose a performance-guaranteed online DBA algorithm based on Online Convex Optimization (OCO), which does not involve static optimizations based on an unreliable estimation of traffic patterns. An OCO algorithm generates a sequence of decisions,

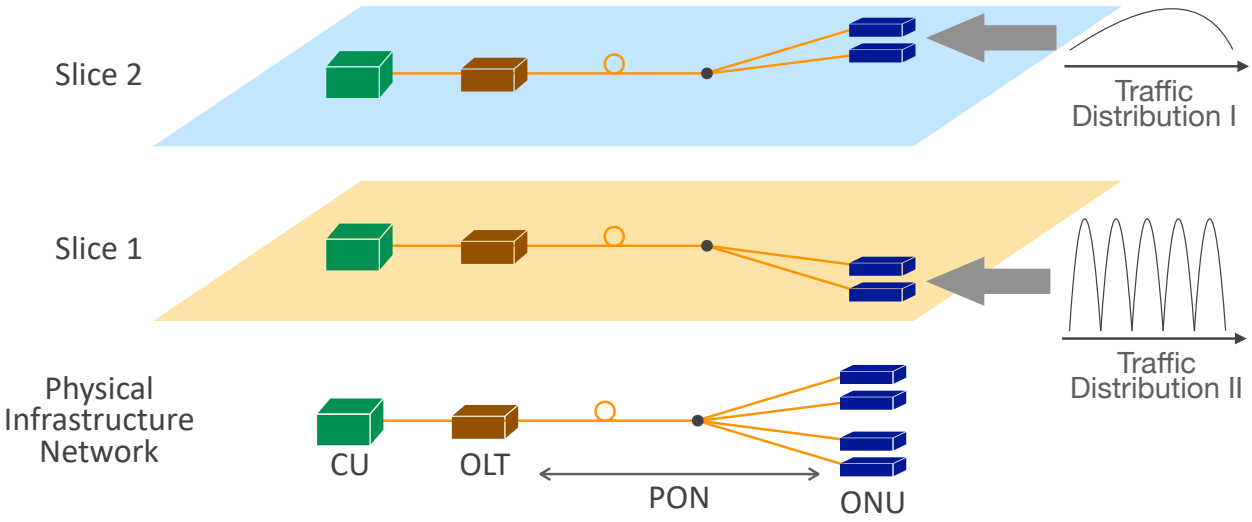

Figure 4.1. PON Access Network Architecture: A physical network is sliced to support different application requirements.

observing the actual past traffic information rather than an estimation of future states. The decision sequence will be adjusted along its runs to minimize the *regret*, the loss incurred by diverting from a decision that is robust over the entire time horizon. Note that such a robust decision only becomes available in retrospect. In the PON context, the OCO-based DBA algorithm decides a window allocation for the next time cycle based on the historical window sizes and *actual* traffic information observed in the previous cycles, so that the resulting allocation becomes robust against the fluctuating traffic. In other words, the algorithm implicitly learns a potential distribution of future traffic and provides a safe window allocation that can accommodate any traffic dynamics that may happen under the distribution. Furthermore, the performance of the algorithm can be theoretically bounded in terms of regret [60]. Our algorithm is more promising than DBA algorithms based on traffic estimation, since its allocation would realize lower latency within a potential traffic range, instead of being optimal only at one estimated traffic amount. Our DBA problem is formulated in such a way as to ensure fairness among ONUs, which guarantees that each ONU receives a minimal size of time window even when it is not heavily loaded. This formulation could

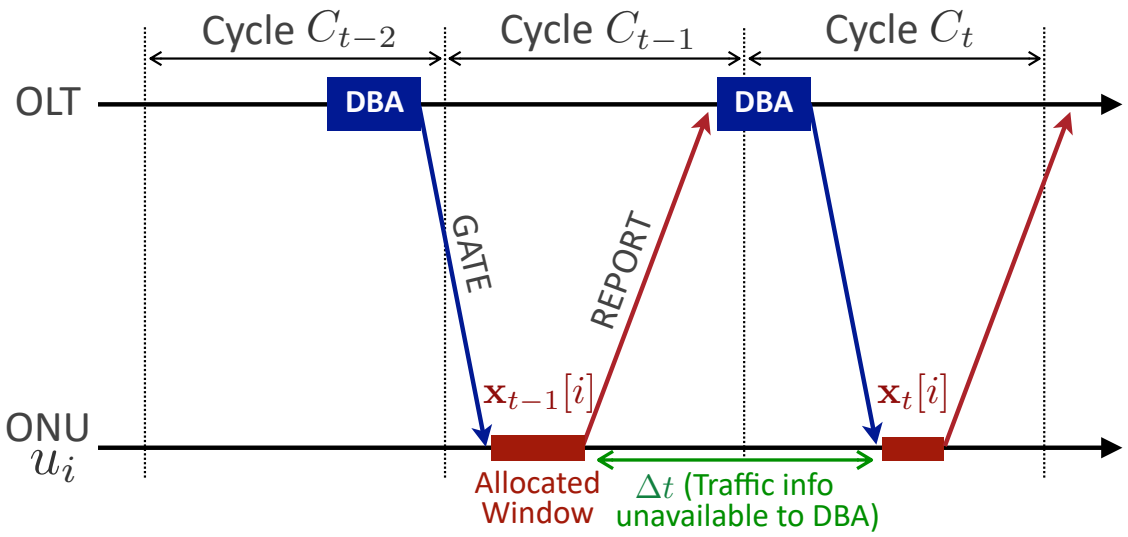

Figure 4.2. Interactions between OLT and ONUs: The OLT decides a bandwidth allocation $\mathbf{x}_t[i]$ to each ONU u_i based on the demand request included in a REPORT message.

be seen as an extension of the work in [43] to a dynamic setting. The simulation results show that our OCO-based DBA algorithm reduces the latency at all ONUs, demonstrating slice-awareness, demand-responsiveness, and allocation fairness.

4.2 PON Access Network Model

A Passive Optical Network (PON) consists of an Optical Line Terminal (OLT) and Optical Network Units (ONUs). Since all the ONUs share one optical cable connecting them to the OLT, they need to be coordinated to share the bandwidth for upstream traffic. Time Division Multiplexing (TDM) PON realizes the sharing by allocating a time window to each ONU in each logical time cycle.

Figure 4.2 illustrates the interaction of an OLT and an ONU $u_i \in U$. An OLT has a Dynamic Bandwidth Allocation (DBA) algorithm that decides the allocation to ONUs. In EPON or 10G EPON standards, an ONU sends a REPORT message that includes a demand request for its next turn. The request is generated by a traffic estimation algorithm at ONUs, which may consider the amount of currently buffered traffic and the future traffic pattern.

Receiving the request, the OLT decides the actual bandwidth allocation for the ONU and informs the ONU of the assigned time windows through a GATE message.

In this paper, we assume that an OLT divides the time horizon into multiple logical time cycles $\{C_t\}$ $(t = 1, ..., T)$ and solves the bandwidth allocation for each time cycle. Note that each ONU receives its turn to send out upstream traffic within the time cycle. With the notation, the amount of bandwidth allocated to an ONU $u_i \in U$ at a cycle C_t is denoted as $\mathbf{x}_t[i]$.

4.3 Problem Statement

Our DBA problem is formulated to provide *slice-aware*, *demand-responsive*, and *fair* bandwidth allocation for ONUs. The slice awareness indicates the capability to satisfy additional delay requirements imposed by some slices. The demand responsiveness is defined as the ability to adjust window sizes based on the amount of traffic load at ONUs. Furthermore, the bandwidth allocation is fair when some ONUs with huge traffic do not dominate a cycle; i.e., each ONU receives an opportunity to send a minimal amount of its queued traffic within each cycle.

The concept of *proportional fairness* is introduced to formulate the DBA problem incorporating the three properties. A vector $\mathbf{x}^*$ is said to be $(\mathbf{w}, 1)$-proportionally fair if the proportions of value differences in any other allocations $\mathbf{x} \in X$ sum to negative:

$$\sum_i \mathbf{w}_i \frac{\mathbf{x}[i] - \mathbf{x}^*[i]}{\mathbf{x}^*[i]} \leq 0 \ (\forall \mathbf{x} \in X). \tag{4.1}$$

It is known that a maximizer $\mathbf{x}^*$ for the objective function of the following form over a convex space $X = \{\mathbf{x}\}$ satisfies the proportional fairness [43].

$$\sum_i \mathbf{w}_i \log \mathbf{x}[i]. \tag{4.2}$$

The intuition behind the relationship between the objective function in Eq. (4.2) and the fairness concept comes from the decreasing increment of logarithmic functions. Let us consider two points x_1 and x_2 such that $x_1 < x_2$. When evaluating the increment of a logarithmic function h with the same increment δ from each point, $h(x_1 + \delta) > h(x_2 + \delta)$ always holds.

In our context, the increment of our objective function, which represents the utility of a specific bandwidth allocation, becomes larger when providing more bandwidth to an ONU that is assigned a smaller bandwidth window. This property enables us to avoid the domination of a time cycle by a small number of ONUs.

The DBA problem is to determine a time partition $\mathbf{x}_t = (\mathbf{x}_t[i])_{i=1,\ldots,|U|}$ of a cycle C_t, where each element in the partition corresponds to a window size allocated to an ONU $u_i \in U$. Let $C \in \mathbb{R}_+$ denote the fixed cycle duration where every ONU receives a fraction of the cycle. An ONU slice $S_j \subseteq U$ is defined as a subset of ONUs that are running on a network slice j, and the prioritization weight p_j indicates the sensitivity of the slice S_j to delay.

Assuming the availability of accurate information regarding the amount $\mathbf{b}_t = (\mathbf{b}_t[i])_{i=1,\ldots,|U|}$ of data queued in every ONU u_i at the starting time of its window in C_t, the optimization problem, which maximizes the total weighted utility of allocated windows, is defined as follows:

$$\text{Maximize } f_t(\mathbf{x}_t) = \sum_i \mathbf{b}_t[i] p_{j:u_i \in S_j} \min\{\log(\mathbf{x}_t[i] + 1), \log(\mathbf{b}_t[i] + 1)\} \tag{4.3}$$

$$\text{subject to } \mathbf{x}_t^\top \mathbf{1}_N \leq C - \sum_i \mathbf{d}[i], \tag{4.4}$$

where $\mathbf{d}$ is a vector of guard window sizes that prevent collisions when switching ONUs.

The min function prevents the utility function from increasing by overallocation. The maximum utility that an ONU u_i can experience at a cycle C_t must be bounded by the allocation that empties its entire buffer ($\mathbf{x}_t[i] = \mathbf{b}_t[i]$). When the allocation is less than the emptying allocation, the utility of the ONU should be represented as an increasing function

over the allocation $\mathbf{x}$. To represent this property, the objective cuts off the increase in the utility by taking a minimum between two logarithm values.

Since the constraint in Eq. (4.4) defines a scaled simplex of $\mathbf{x}_t$, the decision space X is a convex set. The objective function f has the exact same form as the proportionally fair optimization discussed above, recognizing the pair of variables $\mathbf{b}_t[i]$ and $p_{j:u_i \in S_j}$ as a weight $\mathbf{w}_i$.

In addition, the fairness is defined with respect to two weights $\mathbf{b}$ and p that indicate the load on ONUs and the prioritization weight of slices, respectively. Therefore, the maximization with these weights realizes the slice awareness and demand responsiveness, since ONUs with more traffic load and/or with more stringent delay requirements would receive more bandwidth allocations.

4.4 Performance-Guaranteed Dynamic Bandwidth Allocation Based on OCO

4.4.1 OCO-based DBA Algorithm

With the assumption of an oracle to report the accurate amount of queued data $\mathbf{b}_t$, the defined DBA problem is easy to solve, as it has a concave objective function and a convex variable space. However, the practical difficulty of DBA problems arises from the fact that it is impossible to obtain the actual data amount $\mathbf{b}_t$ at the starting time of the window allocated to each ONU. This is because the demand request in a REPORT message is computed based on the data amount at the time when the message is issued. Furthermore, a prediction of future traffic often differs from the actual amount.

The unavailability of the actual traffic amount hinders an optimal bandwidth allocation. Most existing DBA algorithms try to solve the problem by estimating the future traffic amount with statistical or machine learning methods. Nevertheless, the static optimization relying on such estimates may demonstrate high variance in its performance, as the

estimation is not an easy task in general even with recent machine learning techniques. Furthermore, such estimation will be more complex when considering the high user mobility in access networks. Hence, it seems desirable to use an allocation algorithm that gradually adjusts its allocation strategy based on the past traffic patterns to obtain an optimal allocation robust against fluctuating traffic over time, instead of changing an allocation at every time cycle to make it the exact optimum for an estimated traffic amount. In this sense, we want to design an allocation algorithm that is numb to traffic changes over a short time span.

Our proposal is to compute the window size $\mathbf{x}_t$ from the actual traffic from C_0 to C_{t-1} by an Online Convex Optimization (OCO) algorithm, instead of solving the static optimization problem with the estimated amount $\hat{\mathbf{b}}_t$. Since the exact amount of queued data at the previous cycle C_{t-1} is observed after the window allocation $\mathbf{x}_{t-1}$, our algorithm can use the actual data $\mathbf{b}_{t-1}$ to decide an allocation for the next cycle C_t.

By defining a convex version of the objective, $g_t(\mathbf{x}_t) \triangleq -f_t(\mathbf{x}_t)$, the maximization problem of f_t is converted to a minimization problem of a convex function g_t over $\mathbf{x}_t$ with the same constraint in Eq. (4.4). We use Projected Gradient Descent (PGD) [60] to solve this problem. At C_{t-1}, the accurate amounts of queued data are collected at the OLT via REPORT messages (See Figure 4.2). Thus, the actual objective function at the previous cycle g_{t-1} becomes available (Eq. (4.4) with $\mathbf{b}_{t-1}$). Note that this approach is different from the traditional use of REPORT messages in that ONUs report the actual traffic amount after the allocated window at C_{t-1} starts. PGD computes the next window size $\mathbf{x}_t$ by

$$\mathbf{x}_t \leftarrow \Pi_X \left(\mathbf{x}_{t-1} - \eta_{t-1} \nabla g_{t-1}(\mathbf{x}_{t-1}) \right), \tag{4.5}$$

where $\Pi_X(\mathbf{x})$ is a projection bringing $\mathbf{x}_t$ back to the space X, and η_{t-1} is a diminishing learning step size. Since the allocation for the next round is defined as a function of the previous allocation, the recursive relation can be represented as $\mathbf{x}_t \leftarrow \mathrm{PGD}(\mathbf{x}_0, ..., \mathbf{x}_{t-1})$, which was initially desired.

4.4.2 OCO Performance Bound

We are interested in bounding the average performance of a DBA algorithm over time rather than obtaining the exact optimum allocations at each cycle, since the exact optimum solution is not feasible without an oracle reporting the exact future traffic amount $\mathbf{b}_t$. The *regret* of an algorithm A quantifies the discrepancy between a sequence of allocations by A and a hindsight optimum allocation (the most robust allocation over all cycles till C_T):

$$\text{regret}_T(A) = \sup_{\{g_1,\ldots,g_T\}\subseteq G} \left\{ \sum_{t=1}^{T} g_t\left(\mathbf{x}_t\right) - \min_{\mathbf{x}\in X} \sum_{t=1}^{T} g_t(\mathbf{x}) \right\}.$$

Note that the supremum over all possible sequences of objective functions implies that the regret bound considers adversarial scenarios, which is, in our case, a situation where the actual traffic pattern is extremely irregular.

The gradient descent-based solution (PGD), which we use, is known to have $O(\log T)$-bound for the regret with the convergence rate of $1/\sqrt{T}$ for a convex objective function over a convex variable range.

4.5 Experiment and Discussion

Simulations are conducted in a network with three slices: S_0, S_1, and S_2. The network hosts ten ONUs, and each slice has a different number of ONUs as follows: $\{u_0, u_1, ..., u_4\} \in S_0, \{u_5, u_6, u_7\} \in S_1$, and $\{u_8, u_9\} \in S_2$. Among the ten ONUs, ONUs u_0, u_3, u_6, and u_9 are heavily loaded with a traffic generation function based on the Poisson distribution Pois($\lambda = 10$), while the traffic for the other ONUs is generated based on Pois($\lambda = 1$). Unless otherwise specified, the slice prioritization weight p_j is set to 1.0 for all slices. It is only changed when evaluating the slice-awareness.

4.5.1 Performance Comparison: Typical DBA Algorithms

The proposed method is compared with typical generic DBA algorithms: MAXWIN and AVGPRED.

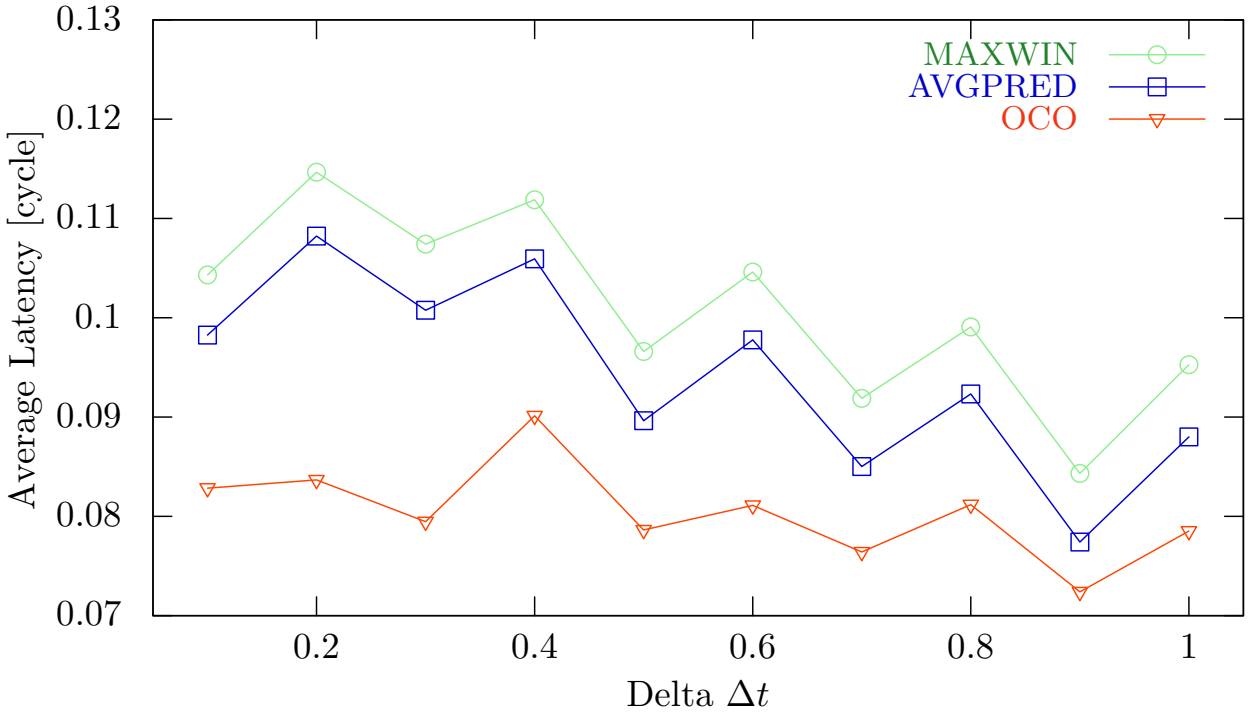

Figure 4.3. Average Latency at ONUs with respect to the cycle size $C = 1$.

MAXWIN

allocates either the predefined maximum window size m or the traffic amount queued in ONUs, which is reported in the previous REPORT message. Thus, this algorithm is quite responsive to the demand from each ONU. While both $m = 0.2$ and 0.4 are tested in the simulations, only the result with $m = 0.2$ is discussed below, since their performances are similar.

AVGPRED

assigns the average of the actual traffic queued at each ONU in past cycles. When computing the average at round C_t, the algorithm uses the past traffic amounts from C_{t-1} to C_{t-h}, where h is a given time horizon. The performance is shown with a horizon h that provided the best result among different h's ($h \in \{10, 100, 1000\}$). This algorithm stays optimum for every

static problem defined with a predicated traffic amount. Note that an allocation vector can be projected back to the solution space to make it a feasible solution, preserving the relative allocation ratio among ONUs.

4.5.2 Experimental Results

We discuss our simulation results from the three aspects desired for a DBA algorithm; namely, demand-responsiveness, slice-awareness, and allocation fairness. In addition, an interpretation of the OCO performance guarantee in our problem is briefly summarized at the end.

Demand-responsiveness

Figure 4.3 illustrates the average latency versus the delta value (Δt) depicted in Figure 4.2, during which additional traffic unknown to a DBA arrives. This value implies the amount of traffic that was not reported to the OLT and can be added to the queues of ONUs before the next round. The y-axis is the average latency of each traffic unit from 50 simulation runs. Δt and the average latency are represented with respect to the cycle size $C = 1$.

The proposed OCO-based algorithm realizes the lowest queuing delay, on average, among the three methods. This result indicates that our solution reduces the average latency even though it does not provide the optimum for every single static optimization. This is because the OCO-based approach gradually adjusts its solution to an allocation that is robust against underlying fluctuations of traffic.

The general decreasing trend of MAXWIN and AVGPRED in the average latency could be explained by excessive adjustment towards traffic amount in each time cycle. As Δt increases, the total amount of unreported traffic becomes more random. Therefore, the negative impact of an unmatched bandwidth allocation will be smaller in scale *on average*. In contrast, the impact stays notable when Δt is smaller. It is worth mentioning that this

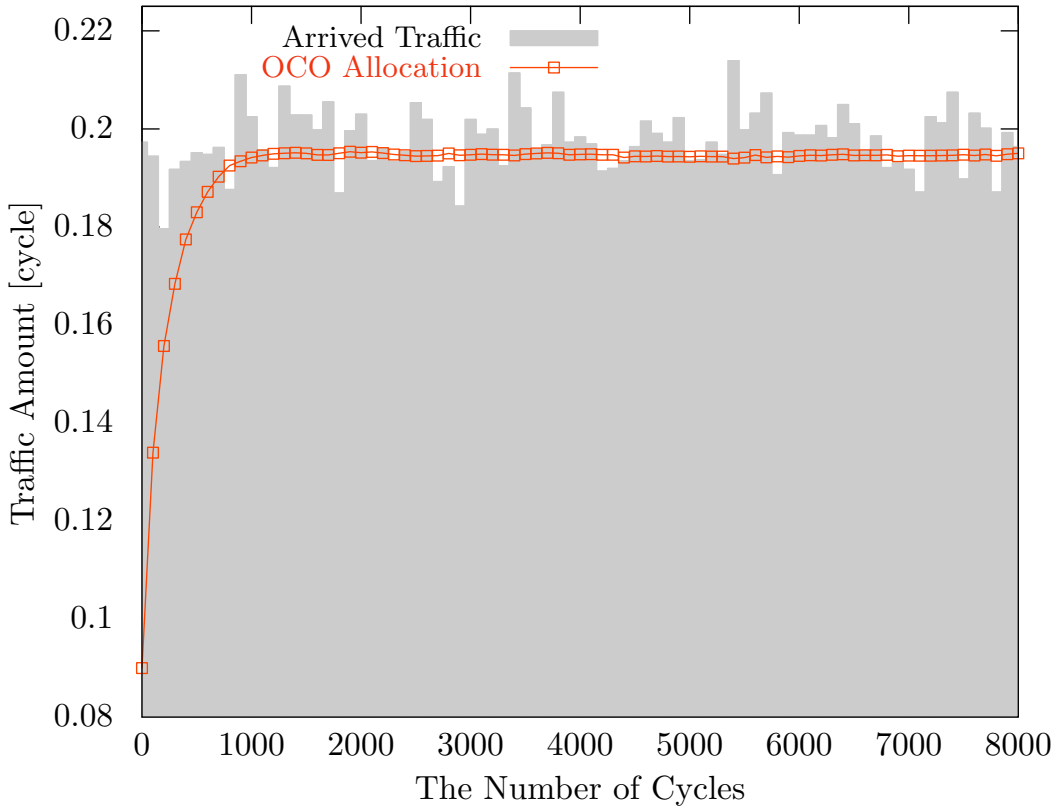

Figure 4.4. Transition of the Allocation to ONU u_0: The proposed DBA algorithm adjusts its allocation over time to provide a robust bandwidth allocation, considering the past traffic pattern.

behavior does not imply the usefulness of MAXWIN and AVGPRED with a larger Δt. This is because it is reasonable to assume the duration between a GATE message and the previous REPORT message (Δt) is within one time cycle C, and our OCO approach performs better in the range.

Figure 4.4 shows the learning process of the allocation to ONU u_0 by the proposed algorithm along with the observed actual traffic pattern (grey bars). The result indicates that the OCO-based algorithm quickly adjusts its allocation to the traffic pattern at earlier cycles and continues fine-tuning at later cycles. The result also indicates the convergence of our OCO-based allocation over time.

Table 4.1. Allocated window sizes with different slice prioritization weights: The slice weight p_j allows delay-sensitive slices to be allocated larger window sizes. (Unit: cycle)

	Latency-sensitive Slice ($p_j = 1.2$)	Standard Slice ($p_j = 1.0$)
Heavily-loaded ONU	0.2642	0.1657
Normally-loaded ONU	0.0569	0.0169

Table 4.2. Average Latency at Each ONU and Standard Deviation σ_U of the Delays in All ONUs in U (Unit: cycle).

	ONU 0	ONU 1	ONU 5	ONU 6	σ_U
OCO	0.0351	0.0172	0.0169	0.0352	0.0094
MAXWIN	0.0230	0.0304	0.0302	0.0231	0.0039
AVGPRED	0.0231	0.0306	0.0310	0.0230	0.0039

Slice-awareness

Table 4.1 indicates the difference in the allocated window sizes depending on the difference among network slices with different delay-sensitivity. The slice prioritization weight p_j was adjusted to represent the specific delay requirements. In particular, assuming that slice S_3 hosts a delay-sensitive network service, the weight p_3 of slice S_3 was set to 1.2, while the weights for the other slices are 1.0. It is observed that the ONUs on S_3 receive more bandwidth. For example, the heavily-loaded ONU u_9 on S_3 is allocated 0.2642 cycles at round C_{10000}, while the heavily-loaded ONUs on S_1 and S_2 receive 0.1657 cycles. This result empirically verifies the possibility to accommodate different types of network slices with diverse requirements through the tuning of the parameter p_j.

Fairness

Figure 4.5, which illustrates the converged allocation to ONU u_0 at C_{10000} by each method, indicates that the heavily-loaded ONUs do not obtain exclusive possession of the bandwidth

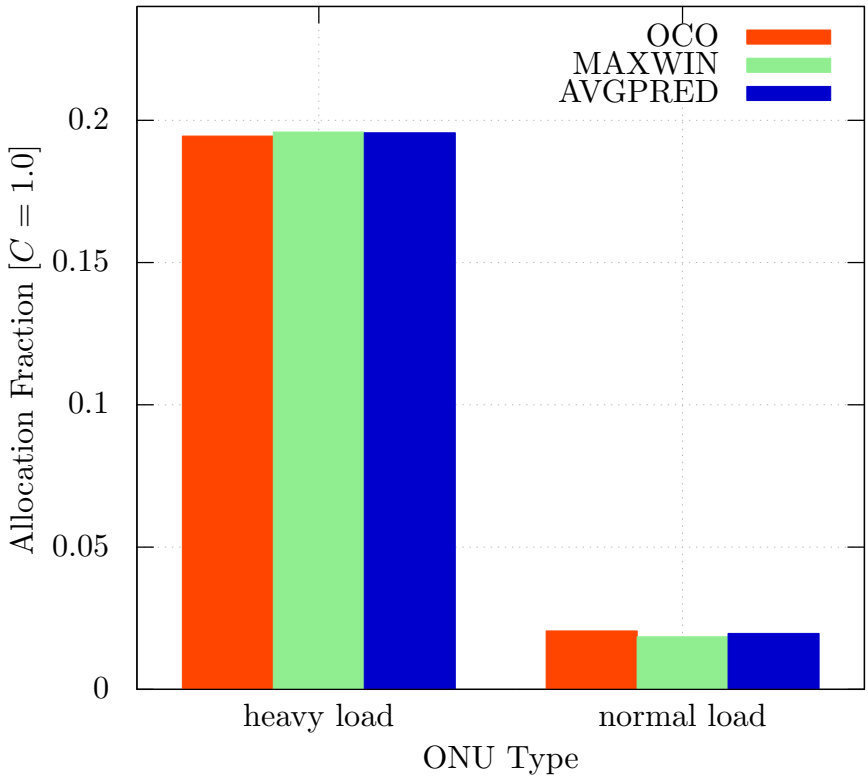

Figure 4.5. Allocation Depending on the Traffic Load at C_{10000}: A heavily-loaded ONU receives more bandwidth.

resource. This result confirms the theoretical discussion on the proportional fairness achieved with a logarithmic objective function.

Table 4.2 summarizes the average latency in the sampled ONUs (u_0, u_1, u_5, and u_6) and the standard deviation σ_U of the delays across all ONUs in U. The standard deviation of the proposed OCO-based algorithm is relatively higher than the other methods. While this implies the prioritization of some ONUs, the standard deviation stays in the same order (10^{-3}). This fact indirectly shows that the proposed algorithm maintains the fairness, even though it allocates more resources to some specific ONUs, so that the latency levels are maintained across all ONUs.

It also shows that the difference in the allocations is quite small (on a scale of 10^{-3}), even though the resulting difference in the average delay by the methods is notable. This fact

implies that the convergence guarantee of OCO techniques provides an additional benefit of a stable robust solution that is gradually updated over time.

4.6 Conclusion

This paper proposes an Online Convex Optimization (OCO) based solution to the Dynamic Bandwidth Allocation (DBA) problem. Our algorithm is aimed at realizing a bandwidth allocation that is slice-aware, demand-responsive, and fair among ONUs in PON access networks, formulating the DBA problem based on the concept of proportional fairness with appropriate weight parameters. The use of the OCO scheme enables our allocation solution to be robust against fluctuations of traffic, which eventually results in the reduction of the average latency over time. The simulation results indicate that the proposed solution mitigates the latency compared to other typical allocation approaches. Furthermore, the results infer the effectiveness of a robust allocation over time for access networks with dynamic traffic patterns, in contrast to the optimum solutions at static optimizations formulated with an estimated traffic pattern.

CHAPTER 5

CLUSTER LEADER ELECTION PROBLEM FOR DISTRIBUTED CONTROLLER PLACEMENT IN SDN

To improve the scalability of Software Defined Networking (SDN), a network can be clustered into subnetworks and managed by multiple distributed SDN controllers, which collaborate through message exchanges with each other. In this chapter[1], we propose a heuristic algorithm for the Cluster Leader Election Problem (CLEP) to decide an effective assignment of distributed SDN controllers in a network consisting of multiple subnetworks (clusters). CLEP deals with the placement problem, considering the metrics between the controllers, such as the distance and connectivity among the controllers in different subnetworks. Moreover, it is shown that some additional properties within each subnetwork, which are discussed in other literature, can be guaranteed by selecting specific clustering methods before the placement.

5.1 Introduction

Software Defined Networking (SDN) has ameliorated the simplicity and programmability of networks by dissociating control logic from forwarding functions. However, SDN is approaching the limitation of its centralized management strategy as the size of SDN networks increases.

SDN with distributed controllers divides a managed network into multiple subnetworks (clusters), and each controller is assigned to manage the switches in its cluster. Because topology and control information of a cluster is encapsulated by each controller, this partially

[1]The content of this chapter is based on the following earlier work.

centralized architecture succeeds in reducing the global information and results in providing scalability to SDN [33].

One of the most important open problems in SDN relates to the placement of a controller in a given network [25]. The initial model of SDN assumed direct connections between all the switches and their controller with southbound communication links that are dedicated only to control messages. Nonetheless, current models suppose that the control messages are sent by communication links of the data plane because of physical and financial reasons (in-band control messages). In these models, a controller is piggybacked onto one of the switches, and the connections between the controller and the dominated switches are realized by paths on the data plane instead of a physical direct link between them. The controller placement problem tries to optimize the metrics between the controller and the switches, such as communication latency [25] and survivability of the logical connections [28], by selecting an appropriate location for the controller.

When deploying multiple controllers, some metrics over clusters (over-cluster metrics) could be considered for the placement in addition to the metrics within a cluster (intra-cluster metrics). For example, the placement is conducted based on the number of disjoint paths between switches and both their initial and backup controller in order to improve the survivability in [45]. Also, the load of each controller, which is commonly defined by the number of switches, is alleviated by balanced placements of controllers [78, 79].

However, these over-cluster metrics only consider the relationship between the switches in a cluster and some controllers. Assuming cooperative operations among distributed SDN controllers, which require state synchronizations, metrics between the controllers would become another important aspect of the multiple controller placement problem.

Hence, this paper proposes a variant of the multiple controller placement problem as the Cluster Leader Election Problem (CLEP). CLEP is defined as a graph optimization problem of minimizing the distance between controllers over which in-band control messages

travel. Additionally, the intractability of CLEP is proved based on reduction to the Set Cover Problem (SCP). Some greedy algorithms for CLEP are also proposed, exploiting an approximation algorithm for SCP. Our simulation compares the performance of these algorithms under different types of graph topologies and clustering methods.

5.2 Assumptions and Our Model

This paper assumes that a communication network is divided into multiple connected subnetworks called clusters. When dealing with the multiple controller placement problem, the clustering method used to divide a whole network into subnetworks influences the result of placement algorithms, and vice versa. When the placement determines the clusters, the solution can achieve the global optimum solution. The placement algorithm based on K-means in [75] indicates the effectiveness of clustering based on the placement of controllers. However, this type of method cannot be applied to all types of metrics discussed in the placement problem because of the non-convexity of some metrics other than distance.

In contrast, when a set of clusters are given as an input for the placement problem, the topological characteristics of each cluster, such as distance and connectivity, can be guaranteed, though it could be possible that a different clustering provides a better solution for an objective function in terms of the global view. Therefore, in our work, a set of clusters is given as input to simplify the dependency between clustering and placement. Section 5.4 shows that the placement problem becomes intractable even with fixed clusters.

The communication nodes in a cluster collaborate with each other through communications within the cluster. When information exchange among clusters is required, a leader node of a cluster on which the controller is piggybacked gathers necessary information in its cluster and broadcasts the information to the adjacent clusters, which are defined as clusters sharing some node. Note that this information exchange is conducted using logical

connections between leader nodes that are realized by in-band control messages on paths in a data-plane network.

The key assumption is that a leader node cannot directly send its information to non-adjacent clusters. The communications with such clusters are conducted by repeated information exchanges between adjacent clusters, although another work [**?**] that considers latency among controllers adopts the longest or average distance between all pairs of the controllers. When the size of networks expands, this relay strategy helps leader nodes to reduce the information that they need to maintain. Additionally, the update for a newly joining cluster or leader node becomes simpler because it only requires local arrangements with neighboring leaders.

In our model, some nodes are included in multiple clusters so that all the edges are covered by clusters. However, each node selects one cluster from all the clusters containing the node. When the node receives in-band control messages from clusters other than its cluster, it relays the messages to appropriate neighbor nodes. For example, in Figure 5.1, v_3 and v_4 are included in both Cluster 1 and 2. These vertices can select to become a member of either of these two clusters. Suppose v_3 chooses Cluster 1. When receiving a message labeled as Cluster 2, v_3 simply relays the message to v_4 or v_5.

5.3 Problem Formulation

5.3.1 Placement Metric

The single controller placement problem is first discussed by Heller et al. in [25] as a problem to minimize the distance between switches and a controller. This latency metric is the most common objective function for the controller placement problem and appears in many other works [82, 5]. In contrast, our CLEP problem is defined as the problem of minimizing latency between controllers, while other works try to reduce the latency between switches and a controller.

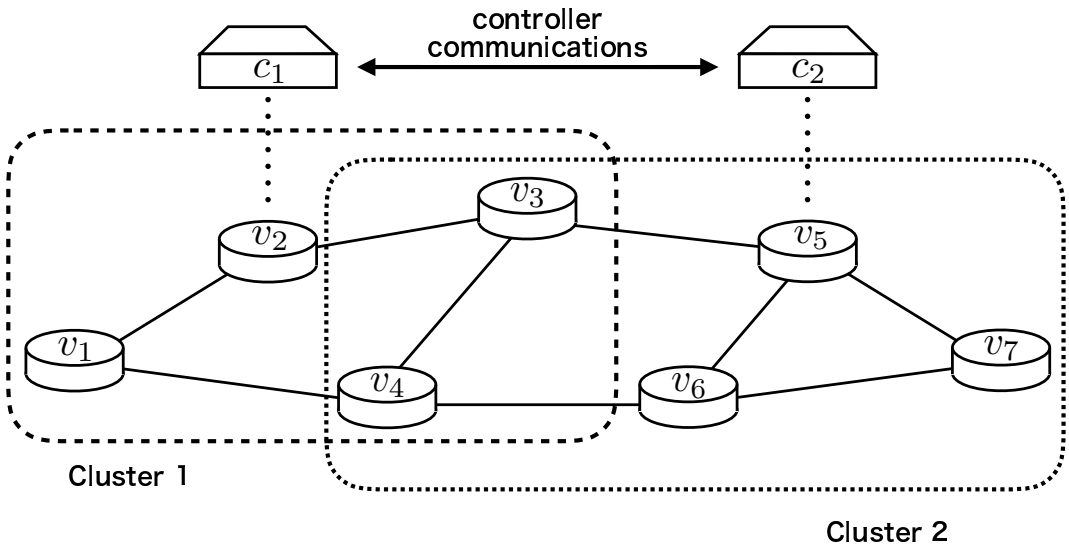

Figure 5.1. In-band SDN model with multiple clusters.

5.3.2 Problem Statement

A communication network is represented as a graph G consisting of vertices V (communication nodes) and edges E that represent connections between vertices. Also, the node groups are named vertex clusters $\mathcal{C} = \{C \subseteq V\}$.

Then, the cluster leader election problem is finding a leader set $H \subseteq V$ that minimizes the distance-based delay metric satisfying the correctness of the leader election defined as follows.

Definition 7. *Correct Leader Set:* When the following is satisfied, the elected leader set $H \subseteq V$ is said to be correct. For every vertex cluster $C \subseteq V$, at least one vertex $\eta \in C$ that is included in the leader set H exists:

$$\forall C \in \mathcal{C}, \exists \eta \in H \subseteq V \text{ such that } \eta \in C. \tag{5.1}$$

This statement means that every cluster needs to have its leader; otherwise, the leader election is not correctly completed. This correctness always holds when a mapping function from a set of leaders to a set of clusters is surjective: $\forall y \in Y, \exists x \in X$ such that $f(x) = y$.

Definition 8. *Cluster Adjacency function:* A cluster adjacency function $\alpha : \mathcal{C} \times \mathcal{C} \longrightarrow \{0, 1\}$ gives 0 iff a given pair of vertex clusters $C \in \mathcal{C}$ and $C' \in \mathcal{C}$ does not share any vertices between them; otherwise, it returns 1.

$$\alpha(C, C') = \begin{cases} 0 & \text{if } C \cap C' = \varnothing \\ 1 & \text{if } C \cap C' \neq \varnothing \end{cases} . \tag{5.2}$$

From the assumption in Section 5.2, the distance based delay for communications among clusters is represented as the sum of distances between each leader $\eta \in H$ and the leaders of the clusters adjacent to the cluster whose leader is η. When $\eta, \eta' \in H$ are the leaders of $C, C' \in \mathcal{C}$ respectively, the delay metric is represented as

$$\sum_{C \in \mathcal{C}} \sum_{C' \in \mathcal{C}} d(\eta, \eta') \alpha(C, C'), \tag{5.3}$$

where $d(v, v')$ $(v, v' \in V)$ is the distance between two vertices. When this metric is minimized by selecting an appropriate set of leaders H^*, the set of leaders is said to be minimum. Thus, the cluster leader election problem is finding the minimum correct leader set in a given weighted graph.

Problem 5. *Cluster Leader Election Problem (CLEP):* Given a graph $G = (V, E)$, a weight function on edges $w : E \longrightarrow \mathbb{R}_+$, a set of vertex clusters $\mathcal{C} = \{C \subseteq V\}$ and a threshold $K \in \mathbb{R}_+$, is there a subset of vertices $H = \{\eta\}$ whose leader function $l : V \supseteq H \longrightarrow \mathcal{C}$ is surjective and the cost defined as follows is less than K?

$$c(H) := \sum_{\eta \in H} \sum_{C \in l(\eta)} \sum_{\eta' \in H} \sum_{C' \in l(\eta')} d(\eta, \eta') \alpha(C, C'), \tag{5.4}$$

where $d(s, t)$ $(s, t \in V)$ is the distance between a pair of vertices s, t in terms of the given edge weight w.

Eq. (5.4) is interpreted as the formal version of Eq. (5.3). The first two summations indicate the sum over all clusters. The first summation provides the total distance metric

for every leader vertex in the leader set H. Because some clusters may designate the same vertex as their leader, the second summation collects the metric for all the clusters whose leader is the given $\eta \in H$. The other two summations represent the sum for all adjacent clusters. The third summation considers all possible destinations for the communication from η, and the fourth summation takes the shared leader into account.

5.4 Intractability of CLEP

In this section, the intractability of CLEP is stated by demonstrating the equivalence of the problem and a well-known NP-complete problem called the *Weighted Minimum Set Cover Problem*.

Problem 6. *Weighted Minimum Set Cover Problem (WMSCP):* Given a set U, a finite family of subsets of U, namely $\mathcal{S} = \{S \subseteq U\}$, a weight function on the subsets $w : \mathcal{S} \longrightarrow \mathbb{R}_+$ and a threshold $K \in \mathbb{R}_+$, is there a subfamily $\mathcal{S}^* \subseteq \mathcal{S}$ such that $\bigcup_{S \in \mathcal{S}^*} S = U$ and the cost defined as follows is less than K?

$$c(\mathcal{S}^*) := \sum_{S \in \mathcal{S}^*} w(S) \ . \tag{5.5}$$

Note that for any cost function w on subsets $S \in \mathcal{S}$, WMSCP is known to be NP-complete [34].

Theorem 5. The Cluster Leader Election Problem is $\mathcal{NP}$-complete.

Proof. Let R_i be a set of clusters such that a cluster C in R_i contains vertex v_i: $R_i := \{C \in \mathcal{C} \mid v_i \in C\}$. Also, $\mathcal{R}$ denotes the collection of all the sets: $\{R_i\}$ $(v_i \in V)$ since R_i is defined for each vertex in V.

Then, with the definition of R_i, the Cluster Leader Election Problem is converted to the following.

Problem 7. *CLEP':* Given a set of vertex clusters $\mathcal{C} = \{C \subseteq V\}$, a finite family of subsets of $\mathcal{C}$, namely $\mathcal{R} = \{R_i \subseteq \mathcal{C}\}$, a weight function on the subsets $w : \mathcal{R} \longrightarrow \mathbb{R}_+$ and a threshold $K \in \mathbb{R}_+$, is there $\mathcal{R}^* \subseteq \mathcal{R}$ that satisfies $\bigcup_{R_i \in \mathcal{R}^*} R_i = \mathcal{C}$ and the cost defined as follows is less than K?

$$c(\mathcal{R}^*) := \sum_{R_i \in \mathcal{R}^*} w(R_i), \tag{5.6}$$

where

$$w(R_i) := \sum_{C \in l(v_i)} \sum_{v_j : R_j \in \mathcal{R}^*} \sum_{C' \in l(v_j)} d(v_i, v_j)\alpha(C, C'). \tag{5.7}$$

The equivalence of CLEP and WMSCP is straightforward from this representation. Because WMSCP is known to be $\mathcal{NP}$-complete for any weight function on the given subsets, CLEP is also $\mathcal{NP}$-complete. □

In the proof of the converted CLEP (Problem 7), the vertex set derived by indices of $R_i \in \mathcal{R}^*$ forms the required leader set that is minimum and correct. The combination of Eq. (5.6) and Eq. (5.7) corresponds to the cost function for the leader set H in Eq. (5.4). Therefore, the threshold K for the cost function $c(\mathcal{R}^*)$ is equivalent to the minimization of the cost of the leader set $c(H)$. This realizes the minimum condition of the leader election. On the other hand, the condition that $\bigcup_{R_i \in \mathcal{R}^*} R_i = \mathcal{C}$ implies the correctness of a leader set. When this condition is satisfied, at least one vertex exists in the leader set derived by R^* for each cluster in $\mathcal{C}$.

5.5 Heuristic Algorithms

In this section, heuristics for CLEP are proposed based on the greedy approximation algorithm for WMSCP [74]. The approximation algorithm for WMSCP selects a subset whose cost efficiency, defined as $\frac{w(R_i)}{|R_i \setminus T|}$, is minimal at each iteration until all the clusters are covered.

Algorithm 4 WMSCP-based greedy algorithm for CLEP

Input: a set of vertex clusters: $\mathcal{C} = \{C \subseteq V\}$, a finite family of subsets of $\mathcal{C}$: $\mathcal{R} = \{R_i \subseteq \mathcal{C}\}$, a weight function on the subsets $w : \mathcal{R} \longrightarrow \mathbb{R}_+$

1: $T \leftarrow \varnothing$, $\mathcal{I} \leftarrow \varnothing$
2: **while** $T \neq \mathcal{C}$ **do**
3: Find $R_i \subseteq \mathcal{C}$ with minimal $\frac{w(R_i)}{|R_i \setminus T|}$
4: $T \leftarrow T \cup R_i$
5: $\mathcal{I} \leftarrow \mathcal{I} \cup i$
6: **end while**

Output: a set of leaders $H = \{v_i \in V\}_{\mathcal{I}}$ $(i \in \mathcal{I})$

5.5.1 WMSCP-based Greedy Algorithm for CLEP

Algorithm 4 describes the steps to elect leaders using the WMSCP algorithm. To obtain the set of leaders, the indices of the selected subsets T are also stored in $\mathcal{I}$ during the set covering execution. This greedy algorithm outputs a set of leaders indexed by $\mathcal{I}$ with the input of a set of clusters $\mathcal{C}$, a family $\mathcal{R}$ of the subsets of $\mathcal{C}$, and a weight function w.

As will be understood, the cost efficiency is determined by the weight function w on the subsets R_i of $\mathcal{C}$ and the number of uncovered elements in the subset. However, the weight function w on R_i defined in Eq. (5.7) requires an exponential number of calculations for all possible combinations of leaders when the number of clusters increases. In general, it is not practical to calculate the exact weight values in an arbitrary network topology. Therefore, our methods try to approximate the values by assuming the worst case scenario and obtaining expected values of the distances between leaders. Note that the computation complexity of the proposed greedy algorithm can be shown by a similar analysis on the complexity of the WMSCP approximation algorithm in [74].

Constant Weight $w_{\mathbf{con}}$

The first type of weight function is a constant weight. This function returns a fixed constant value c regardless of any subset.

$$w_{\text{con}}(R_i) = c. \tag{5.8}$$

Worst Weight $w_{\mathbf{wst}}$

The worst weight function weights R_i with the maximum distance between the leader v_i and all other vertices in all clusters in R_i. This value shows the cost of choosing v_i when one of the clusters in R_i designates a vertex furthest from v_i as its leader.

$$w_{\text{wst}}(R_i) = \max_{C \in R_i} \max_{v_j \in C} d(v_i, v_j). \tag{5.9}$$

Average Weight $w_{\mathbf{avg}}$

Similarly, the average weight function determines a weight for a subset R_i as the average of all the distances from the leader v_i to all other vertices in R_i. This weight is the expected cost to select v_i as a leader vertex.

$$w_{\text{avg}}(R_i) = \frac{1}{\sum_{C \in R_i} |C|} \sum_{C \in R_i} \sum_{v_j \in C} d(v_i, v_j). \tag{5.10}$$

5.5.2 Mapping between Leaders and Clusters

Though the heuristics based on the WMSCP algorithm output a set of vertices H that covers all the clusters with minimal cost, they do not decide which vertex becomes a leader for which cluster when one cluster contains more than one vertex from the set H. Thus, it is necessary to map a cluster to one of the vertices in H to complete the leader election problem.

For simplicity in our mapping process, a leader vertex for a cluster C is decided by uniform random selection from the list of all the possible leader vertices in H. As a result,

each vertex becomes a member of one cluster whose leader gives control information for the vertex and participates in other clusters as a relay vertex. Some improvements in results could be realized by a mapping method that considers topology information.

5.6 Simulation

The proposed greedy algorithms and a centroid algorithm are compared in terms of their performance with respect to the distance between leaders in three types of graph topologies clustered by two clustering methods. The centroid algorithm determines a leader of each cluster so that it can minimize the longest distance from the leader to any other vertices in the cluster. Thus, the centroid algorithm is similar to the common approaches such as K-means [75] for latency reduction except that the clustering is given as an input.

5.6.1 Network Models

The algorithms are compared in their performances in three kinds of graph topologies. For the sake of cycle clustering, each graph is augmented so that it has at least 2-edge connectivity. The assurance of 2-edge connectivity is conducted exploiting Tarjan's bridge detection algorithm [67]. Suppose that the endpoints of a bridge found by the Tarjan's algorithm are v_1 and v_2, and $L(v_i)$ is the set of vertices adjacent to v_i. A new edge is added between a pair of randomly selected vertices from $L(v_1) \cup \{v_1\} \setminus \{v_2\}$ and $L(v_2) \cup \{v_2\} \setminus \{v_1\}$ for each bridge. When the edge creates a self-loop or multi-edge in the graph, the random selection is tried again.

The following types of graph topologies are considered:

Bi-bridged Barbell graph

A bi-bridged barbell graph consists of two complete graphs with n_1 and n_2 vertices and two paths P_1 and P_2 of length l connecting the two complete graphs. It is guaranteed that a pair

of vertices of v_j^1 and v_j^2 is adjacent with one edge of K_{n_j} denoting a lead vertex of path P_i connecting to complete graph K_{n_j} by v_j^i.

Newman Watts Strogatz (NWS) random graph

A Newman Watts Strogatz (NWS) random graph shows the scale-free property and high clustering coefficient as well as the small-world property. The degree distribution of a scale-free random graph follows the power law. Additionally, the high clustering coefficient means that a connected triple of vertices tends to have three edges among them. This graph model first composes a cycle covering all the n vertices and connects each vertex with k nearest neighbors in the cycle. Furthermore, additional edges are spanned between two vertices with the probability parameter p.

Real-world Network Topology

A real-world network topology, called UUNET, is used for original graphs in our simulation. The UUNET is a network of the IP layer with 49 vertices in the United States. The topology information is retrieved from *Internet Topology Zoo* [32].

5.6.2 Cluster Settings

Diam-k Tree (DkT) Clustering

Diam-k Tree (DkT) clustering is based on tree structures with diameter k. Each cluster is induced by the vertices within k hops from a randomly selected root vertex.

Fundamental Cycle (FC) Clustering

Fundamental Cycle (FC) clustering divides a graph into multiple cycle structures using a spanning tree on it. Our simulator first composes a spanning tree from a randomly selected

root vertex based on the Breadth First Search (BFS) algorithm. The addition of one non-tree edge to the spanning tree formulates a cycle, and this cycle becomes a cluster in this method.

5.6.3 Results

Figure 5.2 and 5.3 respectively indicate the average distance from a leader to its adjacent leaders in bi-bridged barbell graphs clustered by D2T and FC clustering. In this simulation, the size of two complete graphs (n_1, n_2) is equally set to $20, 30, 40, 50, 60$, and the length of the paths l between the two complete graphs is $6, 10, 14, 17, 20$, respectively.

The average distances in the NWS random graphs with D1T and FC clustering are shown in Figure 5.4 and 5.5, respectively. The number of vertices n increases from 20 to 100 with the parameters $k = 2$, $p = 0.1$.

Regardless of the difference in diameters between the graph models, the average weight w_{avg} and constant weight w_{con} demonstrate better performance in reducing the average distance for all the graphs. Because of the paths connecting two complete graphs, the diameter of bi-bridged barbell graphs is always equal to $l + 2$. In contrast, NWS graphs have shorter diameters: $E[\text{diam}(G)] \approx 3.0$, which is caused by the random addition of edges based on p.

Due to the high clustering coefficient of complete graphs and NWS graphs, the size of fundamental cycles remains about the same ($E[|C|] \approx 4$) for any graph models. This topological characteristic induces the increase in the number of adjacent cycles along with the growth of the graph size. As a result, the FC clustering tends to increase the average distance to the adjacent leaders. Contrarily, the average size of DkT clusters increases along with the increase in the graph size. Thus, the average distance is moderately augmented in the case of DkT clustering.

Table 5.1 describes the results in the UUNET clustered with both D2T and FC clusterings. In terms of the optimization of the distance, a similar discussion holds for this

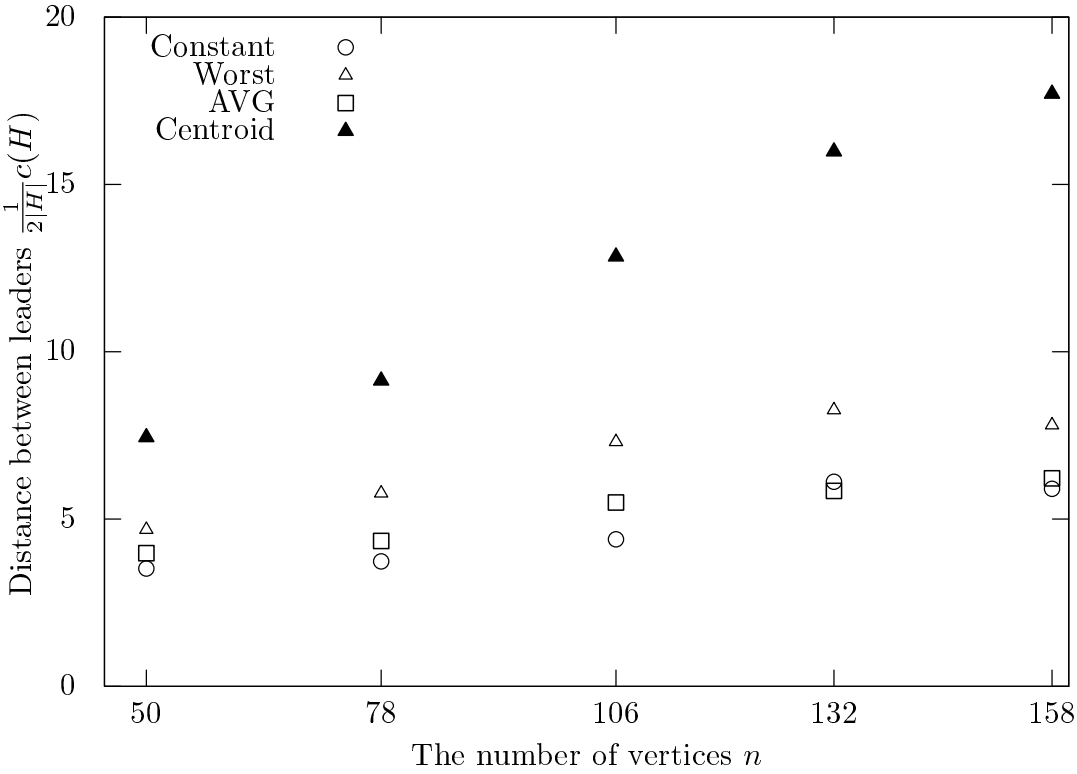

Figure 5.2. Average distance from a leader to its adjacent leaders in bi-bridged barbell graph clustered by D2T clustering.

real-world topology. Since the diameter of real-world communication networks tends to be larger than NWS, the results are similar to the case of bi-bridged barbell graphs.

Figure 5.6 illustrates the comparison of the algorithms and the optimum solution obtained by an exhaustive search in lesser bi-bridged barbell graphs with 16, 18, 20, and 22 vertices. This result implies that the proposed algorithms do not provide solutions that are exceptionally divergent from the optimum solution.

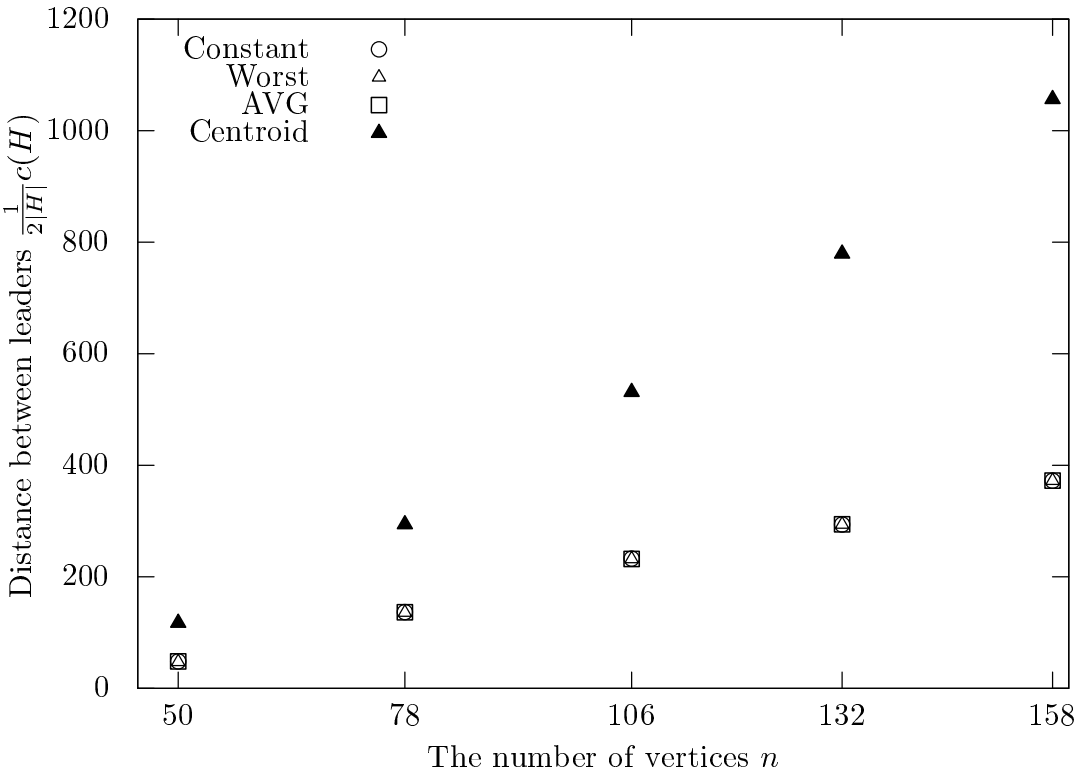

Figure 5.3. Average distance from a leader to its adjacent leaders in bi-bridged barbell graph clustered by FC clustering.

Table 5.1. Average distance from a leader to its adjacent leaders in UUNET with clustered by D2T and FC clustering.

Algorithm	D2T	FC
Constant	7.97	8.53
Worst	8.34	9.22
AVG	**7.45**	**8.16**
Centroid	13.33	20.85

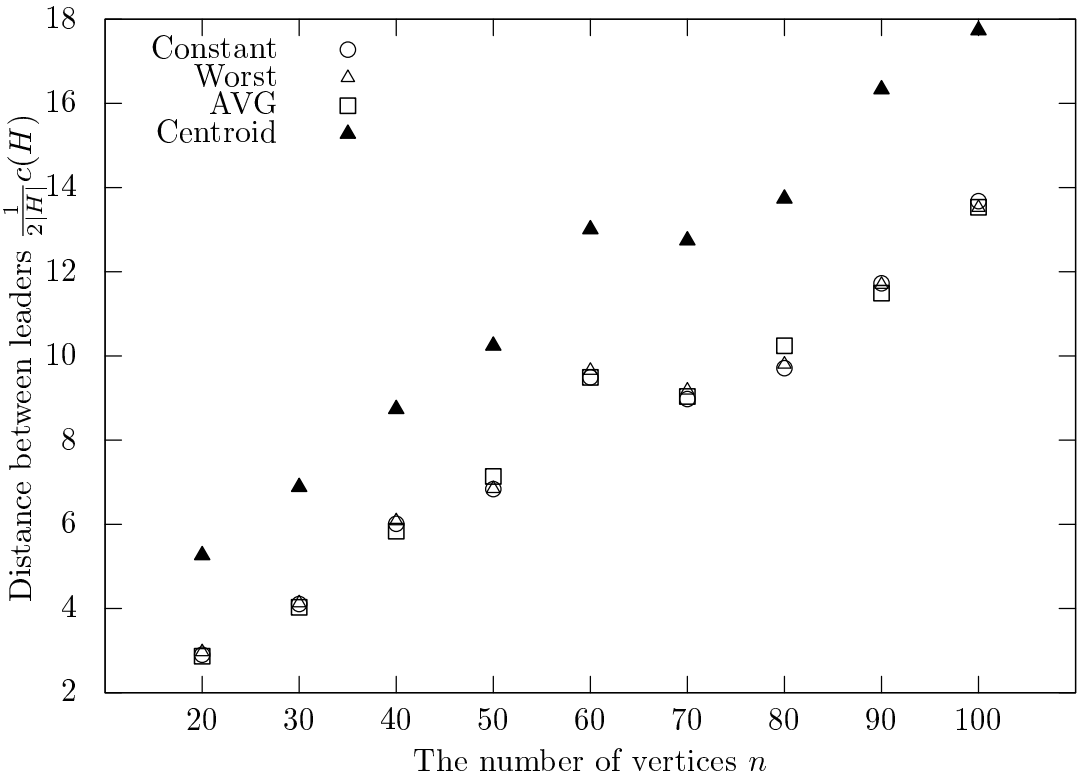

Figure 5.4. Average distance from a leader to its adjacent leaders in NWS random graph clustered by D1T clustering.

5.7 Discussions

5.7.1 Guarantee on the Bound of Intra-metrics

Our greedy algorithms to decide a set of leaders do not consider the metrics within each cluster. However, some metrics such as latency and survivability could be guaranteed based on the property of clustering methods rather than their leader election process.

The latency, which is defined by the distance to a controller from switches, could be upper bounded by analyzing the diameter of clusters. It is obvious that the Diam-k Tree clustering always provides the upper bound for the latency between switches and their controller with length $2k$. When a network is partitioned by the fundamental cycle clustering with BFS, the size of the largest cycle becomes $2 \times \mathrm{diam}(G)$.

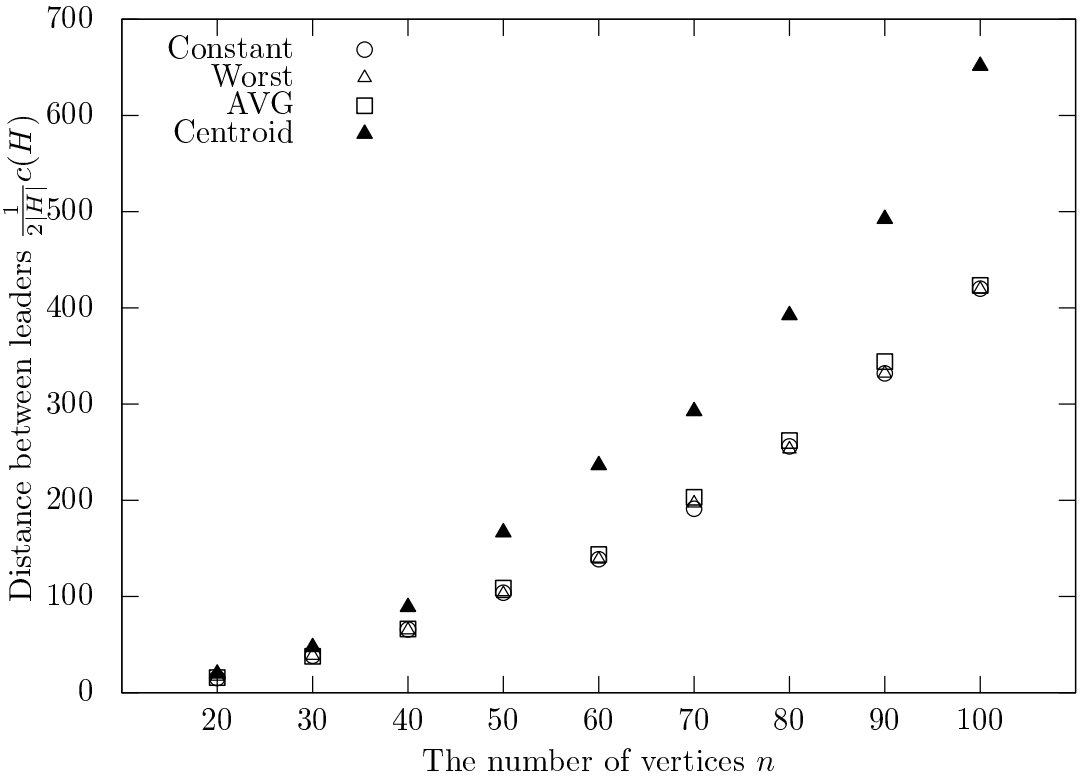

Figure 5.5. Average distance from a leader to its adjacent leaders in NWS random graph clustered by FC clustering.

In addition to latency, survivability is also determined by clustering methods. Because the fundamental cycle clustering divides the entire network into 2-connected clusters, the number of disjoint paths between a controller and switches is equal to 2. In contrast, the Diam-k Tree clustering cannot provide any backup paths for single link failures in a cluster.

5.7.2 Combining Multiple Clusters

As discussed in Section 5.2, the clustering is predetermined as the input before the placement occurs. When the number of clusters is required to be smaller than K in the input, like the work in [75], combining some clusters makes our algorithms satisfy the requirement.

To avoid forming a giant cluster, it is better to select the smallest cluster and combine it with the smallest adjacent cluster. After repeating this procedure until the number of

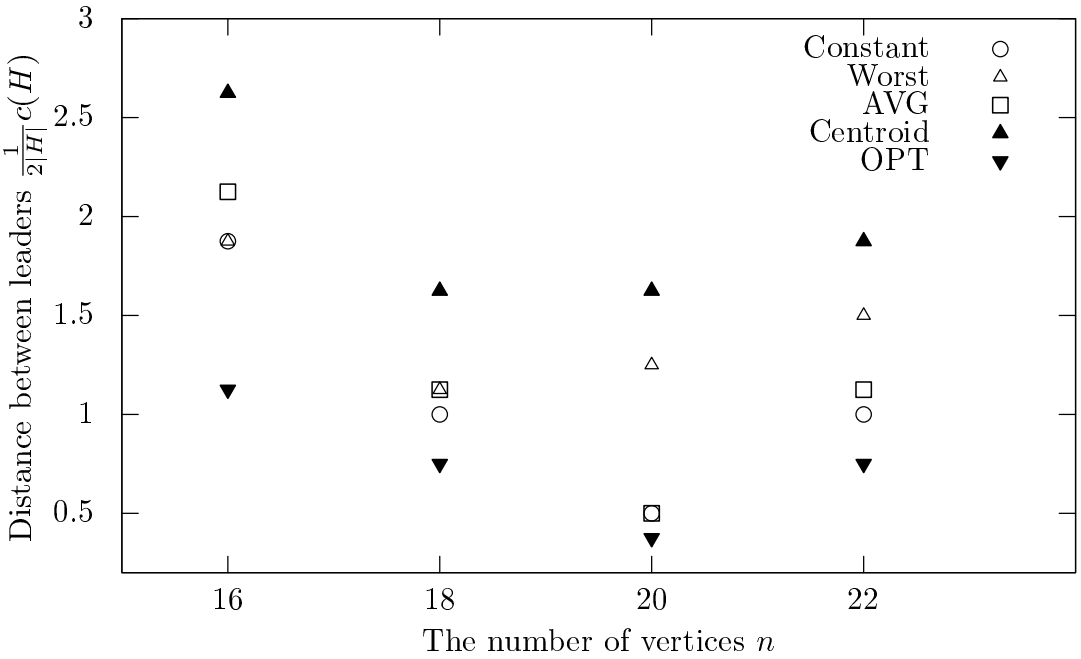

Figure 5.6. Average distance from a leader to its adjacent leaders in bi-bridged barbell graph clustered by D1T clustering.

clusters becomes less than and equal to K, the proposed algorithm can be used to decide the placement. Even in this case, the discussion on the bound of intra-metrics above holds because the sum of the upper bounds on intra-metrics of combined clusters gives the upper bound for the new cluster at each combining step. It is obvious that the diameter of the new cluster is the sum of diameters of two clusters in the worst case.

5.7.3 NP-completeness under Other Assumptions

Though the objective of our Cluster Leader Election problem is the minimization of the distance between controllers, any CLEP aiming at optimizing different metrics between controllers becomes NP-complete. In the proof discussed in Section 5.4, the only part depending on the distance metric is the weight function on the leader set of Eq. (5.6). However, the NP-completeness still holds with other functions because WMSCP is known to be NP-complete for any weight functions.

Furthermore, CLEP stays NP-complete even after relaxing the assumption on the multiple memberships of a communication node to some clusters. In other works on the clustering methodology, the membership of a node can be restricted to exactly one cluster to assure the disjointness of clusters. Though the size of a set R_i remains equal to 1 under this assumption, the completeness proof holds.

5.8 Conclusion

In this paper, we formulated the Cluster Leader Election Problem (CLEP), generalizing the multiple controller placement problem of allocating a controller to each cluster while minimizing the distance metric between the controllers. CLEP is proven to be NP-complete by reducing it to Weighted Minimum Set Cover Problem (WMSCP). Exploiting the well-known approximation algorithm for WMSCP, algorithms with three kinds of weights for greedy choices have been proposed. The simulation results indicate the effectiveness of the proposed greedy algorithms using the average distance weight and constant weight.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

This dissertation discusses how Machine Learning (ML) and combinatorial optimization techniques can realize effective resource coordination for network slicing under complex interdependency among network entities and uncertainty of future traffic and available resources. In Chapter 2, we reveal the role of cycle dependency structures in the context of network survivability and propose a dependency adjustment algorithm that enhances the network-wide reliability by restructuring the interdependency between the network function and infrastructure layers. In addition to the network protection method against the cascading failures, a recovery resource allocation algorithm is devised based on Deep Reinforcement Learning (Deep RL) in Chapter 3. The allocation algorithm learns the implicit importance of the failed infrastructure nodes and determines an optimal resource allocation order to maximize the interim network service functionality. Chapter 4 focuses on the uncertainty of future traffic while deciding an optimal bandwidth allocation to network slices in optical access networks. The proposed Online Convex Optimization (OCO)-based algorithm gradually updates the bandwidth allocation to find an appropriate bandwidth that reduces the overall latency of upstream traffic in access networks. In Chapter 5, we propose a placement algorithm for distributed SDN controllers, which are key enablers of end-to-end network slicing. The reduction of communication overheads by an optimal controller placement leads to the improvement of controller responsiveness to resource coordination requests.

6.2 Future Works

6.2.1 Dynamic Bandwidth Allocation for End-to-End Elastic Network Slices

Chapter 4 introduces a learning-based Dynamic Bandwidth Allocation algorithm for Passive Optical Networks (PONs), which is the initial step towards effective resource coordination of a whole network slice from the access to core networks. Our current approach employs an OCO algorithm and the concept of fairness. The use of an OCO algorithm provides a general framework that guarantees performance bounds in terms of regret. Integrating fair allocation into such an algorithm, it is shown that the average delay on each network slice decreases due to more efficient bandwidth utilization, satisfying the isolation among slices. This initial result in the PON-based access networks needs to be extended to the network-wide level. However, the extension would involve a complication of the objective function, which may prevent us from simply adopting a convex optimization technique. In that case, we need to propose an approximately boundable allocation method for combinatorial objectives based on the OCO approach.

6.2.2 Pricing Mechanism for Elastic Network Slices

Another approach to deal with the uncertainty of the future demand of network resources is to shape the future by indirectly guiding the behaviors of slice users through dynamic pricing. Common cloud computing platforms such as Amazon Web Services (AWS) have pricing mechanisms to solve similar problems for their computing resources. It will be effective if network slice providers use their mechanisms to dynamically adjust the prices of elastic network slices. This research task would involve two kinds of estimation problems: estimation of the user-type distribution and estimation of the blocking probability of slice scaling requests. To motivate slice tenants to change their request behaviors, it is important to consider the users' *prior preferences* over the price range. Assuming that the preferences

are represented as their private utility functions, a slice provider needs to estimate the distribution of the types of utility functions of all the potential tenants. Furthermore, it is critical to maintaining the blocking rate of scale-up requests under a certain threshold to guarantee the performance of elastic network slices.

REFERENCES

[1] Abdelwahab, S., B. Hamdaoui, M. Guizani, and T. Znati (2016). Network function virtualization in 5G. *IEEE Communications Magazine 54*(4), 84–91.

[2] Agiwal, M., A. Roy, and N. Saxena (2016). Next generation 5g wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials 18*(3), 1617–1655.

[3] Ahmed, J., J. Chen, L. Wosinska, B. Chen, and B. Mukherjee (2013). Efficient inter-thread scheduling scheme for long-reach passive optical networks. *IEEE Communications Magazine 51*(2), S35–S43.

[4] Andersson, G., P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, J. Sanchez-Gasca, R. Schulz, A. Stankovic, C. Taylor, and V. Vittal (2005, Nov). Causes of the 2003 major grid blackouts in north america and europe, and recommended means to improve system dynamic performance. *IEEE Transactions on Power Systems 20*(4), 1922–1928.

[5] Bari, M., A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed, and R. Boutaba (2013, Oct). Dynamic controller provisioning in software defined networks. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pp. 18–25.

[6] Behfarnia, A. and A. Eslami (2017, July). Error correction coding meets cyber-physical systems: Message-passing analysis of self-healing interdependent networks. *IEEE Transactions on Communications 65*(7), 2753–2768.

[7] Berizzi, A. (2004, June). The Italian 2003 blackout. In *IEEE Power Engineering Society General Meeting, 2004.*, pp. 1673–1679 Vol.2.

[8] Bienstock, D., J. Sethuraman, and C. Ye (2013). Approximation algorithms for the incremental knapsack problem via disjunctive programming. In *arXiv:1311.4563 [cs.DS]*. `https://arxiv.org/abs/1311.4563`.

[9] Buldyrev, S. V., R. Parshani, G. Paul, H. E. Stanley, and S. Havlin (2010, Apr). Catastrophic cascade of failures in interdependent networks. *Nature 464*(7291), 1025–1028.

[10] Chattopadhyay, S., H. Dai, D. Y. Eun, and S. Hosseinalipour (2017, Sept). Designing optimal interlink patterns to maximize robustness of interdependent networks against cascading failures. *IEEE Transactions on Communications 65*(9), 3847–3862.

[11] Checko, A., H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann (2015). Cloud RAN for mobile networks—a technology overview. *IEEE Communications Surveys & Tutorials 17*(1), 405–426.

[12] Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of Operations Research 4*(3), 233–235.

[13] Ciavarella, S., N. Bartolini, H. Khamfroush, and T. L. Porta (2017, May). Progressive damage assessment and network recovery after massive failures. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9.

[14] ETSI (Retrieved on February 10, 2020a). Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework V3.1.1 (2017-12). `https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf`.

[15] ETSI (Retrieved on February 10, 2020b). OSM scope, functionality, operation and integration guidelines (issue 1). OSM White Paper, `https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf`.

[16] ETSI, N. (2014). Gs nfv-man 001 v1. 1.1 network function virtualisation (nfv); management and orchestration.

[17] Fan, J., C. Guan, Y. Zhao, and C. Qiao (2017, May). Availability-aware mapping of service function chains. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9.

[18] Fan, J., M. Jiang, O. Rottenstreich, Y. Zhao, T. Guan, R. Ramesh, S. Das, and C. Qiao (2018, Oct). A framework for provisioning availability of NFV in data center networks. *IEEE Journal on Selected Areas in Communications 36*(10), 2246–2259.

[19] Ferdousi, S., F. Dikbiyik, M. Tornatore, and B. Mukherjee (2017, March). Joint progressive recovery of optical network and datacenters after large-scale disasters. In *2017 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3.

[20] Foukas, X., G. Patounas, A. Elmokashfi, and M. K. Marina (2017). Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine 55*(5), 94–100.

[21] Gallos, L. K. and N. H. Fefferman (2015, Nov). Simple and efficient self-healing strategy for damaged complex networks. *Phys. Rev. E 92*, 052806.

[22] Gao, J., S. V. Buldyrev, H. E. Stanley, and S. Havlin (2012). Networks formed from interdependent networks. *Nature physics 8*(1), 40–48.

[23] Hatem, J. A., A. R. Dhaini, and S. Elbassuoni (2019). Deep learning-based dynamic bandwidth allocation for future optical access networks. *IEEE Access 7*, 97307–97318.

[24] Hawilo, H., M. Jammal, and A. Shami (2019, March). Network function virtualization-aware orchestrator for service function chaining placement in the cloud. *IEEE Journal on Selected Areas in Communications 37*(3), 643–655.

[25] Heller, B., R. Sherwood, and N. McKeown (2012). The controller placement problem. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pp. 7–12. ACM.

[26] Helmy, A. and A. Nayak (2018). Towards more dynamic energy-efficient bandwidth allocation in epons. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.

[27] Helmy, A. and A. Nayak (2020). Energy-efficient decentralized framework for the integration of fog with optical access networks. *IEEE Transactions on Green Communications and Networking 4*(3), 927–938.

[28] Hu, Y., W. Wendong, X. Gong, X. Que, and C. Shiduan (2013, May). Reliability-aware controller placement for software-defined networks. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pp. 672–675.

[29] Ishigaki, G., S. Devic, R. Gour, and J. P. Jue (2019). DeepPR: Incremental Recovery for Interdependent VNFs with Deep Reinforcement Learning. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.

[30] Ishigaki, G., R. Gour, and J. P. Jue (2018, May). Improving the survivability of interdependent networks by restructuring dependencies. In *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6.

[31] Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing 4*(1), 77–84.

[32] Knight, S., H. Nguyen, N. Falkner, R. Bowden, and M. Roughan (2011, october). The internet topology zoo. *Selected Areas in Communications, IEEE Journal on 29*(9), 1765 –1775.

[33] Koponen, T., M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker (2010). Onix: A distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, Berkeley, CA, USA, pp. 351–364. USENIX Association.

[34] Korte, B., J. Vygen, B. Korte, and J. Vygen (2012). *Combinatorial optimization*, Volume 2. Springer.

[35] Kulkarni, S. G., G. Liu, K. K. Ramakrishnan, M. Arumaithurai, T. Wood, and X. Fu (2018). Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, New York, NY, USA, pp. 41–53. ACM.

[36] Lara, A., A. Kolasani, and B. Ramamurthy (2014). Network innovation using openflow: A survey. *IEEE Communications Surveys Tutorials 16*(1), 493–512.

[37] Lee II, E. E., J. E. Mitchell, and W. A. Wallace (2007, Nov). Restoration of services in interdependent infrastructure systems: A network flows approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 37*(6), 1303–1317.

[38] Letaief, K. B., W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang (2019). The roadmap to 6g: Ai empowered wireless networks. *IEEE Communications Magazine 57*(8), 84–90.

[39] Liu, J., Z. Jiang, N. Kato, O. Akashi, and A. Takahara (2016, June). Reliability evaluation for NFV deployment of future mobile broadband networks. *IEEE Wireless Communications 23*(3), 90–96.

[40] Lu Shen, Xi Yang, and B. Ramamurthy (2005). Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks. *IEEE/ACM Transactions on Networking 13*(4), 918–931.

[41] Majdandzic, A., L. A. Braunstein, C. Curme, I. Vodenska, S. Levy-Carciente, H. Eugene Stanley, and S. Havlin (2016, March). Multiple tipping points and optimal repairing in interacting networks. *Nature Communications 7:10850*.

[42] Mazumder, A., C. Zhou, A. Das, and A. Sen (2016). Progressive recovery from failure in multi-layered interdependent network using a new model of interdependency. In C. G. Panayiotou, G. Ellinas, E. Kyriakides, and M. M. Polycarpou (Eds.), *Critical Information Infrastructures Security*, Cham, pp. 368–380. Springer International Publishing.

[43] Merayo, N., P. Pavon-Marino, J. C. Aguado, R. J. Durán, F. Burrull, and V. Bueno-Delgado (2017). Fair bandwidth allocation algorithm for pons based on network utility maximization. *IEEE/OSA Journal of Optical Communications and Networking 9*(1), 75–86.

[44] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). Human-level control through deep reinforcement learning. *Nature 518*(7540), 529.

[45] Muller, L., R. Oliveira, M. Luizelli, L. Gaspary, and M. Barcellos (2014, Dec). Survivor: An enhanced controller placement strategy for improving sdn survivability. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 1909–1915.

[46] Navarro-Ortiz, J., P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler (2020). A survey on 5g usage scenarios and traffic models. *IEEE Communications Surveys Tutorials 22*(2), 905–929.

[47] NetworkX (Retrieved on March 27, 2019). networkx.generators.random_graphs.gnp_random_graph. `https://networkx.github.io/`.

[48] Nguyen, D. T., Y. Shen, and M. T. Thai (2013, March). Detecting critical nodes in interdependent power networks for vulnerability assessment. *IEEE Transactions on Smart Grid 4*(1), 151–159.

[49] Nunes, B. A. A., M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials 16*(3), 1617–1634.

[50] Ordonez-Lucena, J., P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira (2017). Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges. *IEEE Communications Magazine 55*(5), 80–87.

[51] Ouyang, M. (2014). Review on modeling and simulation of interdependent critical infrastructure systems. *Reliability Engineering & System Safety 121*(Supplement C), 43 – 60.

[52] Parandehgheibi, M. and E. Modiano (2013, Dec). Robustness of interdependent networks: The case of communication networks and the power grid. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pp. 2164–2169.

[53] Pourvali, M., K. Liang, F. Gu, H. Bai, K. Shaban, S. Khan, and N. Ghani (2016, Feb). Progressive recovery for network virtualization after large-scale disasters. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–5.

[54] Pérez, G. O., J. A. Hernández, and D. Larrabeiti (2018). Fronthaul network modeling and dimensioning meeting ultra-low latency requirements for 5g. *IEEE/OSA Journal of Optical Communications and Networking 10*(6), 573–581.

[55] Qu, L., C. Assi, K. Shaban, and M. J. Khabbaz (2017, Sep.). A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks. *IEEE Transactions on Network and Service Management 14*(3), 554–568.

[56] Rahnamay-Naeini, M. (2016, Feb). Designing cascade-resilient interdependent networks by optimum allocation of interdependencies. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–7.

[57] Rastegarfar, H., D. C. Kilper, M. Glick, and N. Peyghambarian (2015, Dec). Cyber-physical interdependency in dynamic software-defined optical transmission networks. *IEEE/OSA Journal of Optical Communications and Networking 7*(12), 1126–1134.

[58] Sen, A., A. Mazumder, J. Banerjee, A. Das, and R. Compton (2014, April). Identification of K most vulnerable nodes in multi-layered network using a new model of interdependency. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 831–836.

[59] Shafin, R., L. Liu, V. Chandrasekhar, H. Chen, J. Reed, and J. C. Zhang (2020). Artificial intelligence-enabled cellular networks: A critical path to beyond-5g and 6g. *IEEE Wireless Communications 27*(2), 212–217.

[60] Shalev-Shwartz, S. (2012). Online learning and online convex optimization. *DOI:10.1561/2200000018*.

[61] Shin, D. H., D. Qian, and J. Zhang (2014, July). Cascading effects in interdependent networks. *IEEE Network 28*(4), 82–87.

[62] Skubic, B., J. Chen, J. Ahmed, L. Wosinska, and B. Mukherjee (2009). A comparison of dynamic bandwidth allocation for epon, gpon, and next-generation tdm pon. *IEEE Communications Magazine 47*(3), S40–S48.

[63] Stippinger, M. and J. Kertész (2014). Enhancing resilience of interdependent networks by healing. *Physica A: Statistical Mechanics and its Applications 416*, 481 – 487.

[64] Sturaro, A., S. Silvestri, M. Conti, and S. K. Das (2016, Feb). Towards a realistic model for failure propagation in interdependent networks. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–7.

[65] Su, R., D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu (2019). Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models. *IEEE Network 33*(6), 172–179.

[66] Taleb, T., A. Ksentini, and R. Jantti (2016, November). "anything as a service" for 5g mobile systems. *IEEE Network 30*(6), 84–91.

[67] Tarjan, R. E. (1974). A note on finding the bridges of a graph. *Information Processing Letters 2*(6), 160–161.

[68] Tauch, S., W. Liu, and R. Pears (2015, April). Measuring cascade effects in interdependent networks by using effective graph resistance. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 683–688.

[69] TensorFlow (Retrieved on March 27, 2019). tf.train.AdamOptimizer. `https://www.tensorflow.org/versions/r1.14/api_docs/python/tf/train/AdamOptimizer`.

[70] The Internet Topology Zoo (Retrieved on March 27, 2019a). BT North America. `http://www.topology-zoo.org/dataset.html`.

[71] The Internet Topology Zoo (Retrieved on September 23, 2019b). IBM. `http://www.topology-zoo.org/dataset.html`.

[72] Uzawa, H., K. Honda, H. Nakamura, Y. Hirano, K. Nakura, S. Kozaki, and J. Terada (2020). Dynamic bandwidth allocation scheme for network-slicing-based tdm-pon toward the beyond-5G era. *IEEE/OSA Journal of Optical Communications and Networking 12*(2), A135–A143.

[73] Vassilaras, S., L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos (2017). The algorithmic aspects of network slicing. *IEEE Communications Magazine 55*(8), 112–119.

[74] Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

[75] Wang, G., Y. Zhao, J. Huang, Q. Duan, and J. Li (2016, May). A k-means-based network partition algorithm for controller placement in software defined network. In *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6.

[76] Wang, J., C. Qiao, and H. Yu (2011, April). On progressive network recovery after a major disruption. In *2011 Proceedings IEEE INFOCOM*, pp. 1925–1933.

[77] Xu, M., X. Liu, N. Chand, F. Effenberger, and G. Chang (2017). Flex-frame timing-critical passive optical networks for delay sensitive mobile and fixed access services. In *2017 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3.

[78] Yao, G., J. Bi, Y. Li, and L. Guo (2014, Aug). On the capacitated controller placement problem in software defined networks. *Communications Letters, IEEE 18*(8), 1339–1342.

[79] Yao, L., P. Hong, W. Zhang, J. Li, and D. Ni (2015, June). Controller placement and flow based dynamic management problem towards sdn. In *Communications (ICC), 2015 IEEE International Conference on*, pp. 369–374.

[80] Yousaf, F. Z., V. Sciancalepore, M. Liebsch, and X. Costa-Perez (2019). Manoaas: A multi-tenant nfv mano for 5g network slices. *IEEE Communications Magazine 57*(5), 103–109.

[81] Zhang, S. (2019). An overview of network slicing for 5G. *IEEE Wireless Communications 26*(3), 111–117.

[82] Zhang, Y., N. Beheshti, and M. Tatipamula (2011, Dec). On resilience of split-architecture networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6.

[83] Zhao, Y., M. Pithapur, and C. Qiao (2016, Dec). On progressive recovery in interdependent cyber physical systems. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.

[84] Zhao, Y. and C. Qiao (2017, May). Enhancing the robustness of interdependent cyber-physical systems by designing the interdependency relationship. In *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6.

BIOGRAPHICAL SKETCH

Genya Ishigaki received his BS in Engineering and MS in Engineering from Soka University, Japan, and MS in Computer Science from The University of Texas at Dallas. His research interests lie in next-generation telecommunication networks with an emphasis on developing combinatorial optimization and machine learning techniques to address problems related to resource allocation. He served the Department of Computer Science at UT Dallas as a teaching assistant and research assistant. Genya also taught an undergraduate course on discrete mathematics at UT Dallas for four semesters. During the last year of his PhD program, he worked for TieSet, an early-stage startup company based in Santa Clara, CA, as a research intern to develop an asynchronous Federated Learning system. He received two scholarships during his PhD program from Shigeta Education Foundation and Japan Student Services Organization.

CURRICULUM VITAE

Genya Ishigaki

April 30, 2021

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Email: gishigaki@utdallas.edu

Educational History:

PhD, Computer Science, The University of Texas at Dallas, 2021
MS, Computer Science, The University of Texas at Dallas, 2021
MS, Engineering, Soka University, Japan, 2016
BS, Engineering, Soka University, Japan, 2014

Dynamic Resource Coordination towards Reliable and Flexible Network Slicing
PhD Dissertation
Department of Computer Science, The University of Texas at Dallas
Advisors: Dr. Jason Jue

On composing a resilient tree in a communication network with intermittent connections based on stress centrality
Master Thesis
Department of Information Systems Science, Soka University, Japan
Advisor: Dr. Norihiko Shinomiya

Distributed network flow optimization algorithm based on tie-set control with coloring
Bachelor Thesis
Department of Information Systems Science, Soka University, Japan
Advisor: Dr. Norihiko Shinomiya

Employment History:

Research Assistant, The University of Texas at Dallas, August 2019 – May 2021
Research Intern, TieSet, Inc., May 2020 – December 2020
Teaching Assistant, The University of Texas at Dallas, August 2018 – August 2019
Teaching Assistant, The University of Texas at Dallas, August 2017 – December 2017

Professional Recognitions and Honors:

Scholarship for PhD Study, Shigeta Education Foundation, 2019 – 2021
Outstanding Teaching Assistant Award, Department of Computer Science, UTD, 2019
NSF Student Travel Grant, IEEE Globecom, 2019
Scholarship for PhD Study, Japan Student Services Organization, 2016 – 2019
Travel Grant, NEC C&C Foundation, 2015
Top Graduate, Department of Information Systems Science, Soka University, Japan, 2014
Student Research Award, IEICE ES Society, 2014
Global Citizenship Program (Honors Program), Soka University, Japan, 2010 – 2014

Professional Memberships:

Institute of Electrical and Electronics Engineers (IEEE), 2014 – present
IEEE Tokyo Young Professionals (YP) Committee, 2015 – present