End-to-End Network Slicing Design Policy in 5G Networks

Wang, Ranyin

Awarding institution:
King's College London

End-to-End Network Slicing Design Policy in 5G Networks

Ranyin Wang

Supervisor: Prof A. Hamid Aghvami

Department of Engineering

King's College London

A thesis submitted to King's College London in fulfillment

the requirements for the Degree of

Doctor of Philosophy

February 2023

Acknowledgements

The PhD study over the last four years has been a memorable and precious experience for me. Here, I express my sincere gratitude to all the people who have helped and encouraged me along my path.

First, I would like to express my genuine appreciation to my supervisor Prof Hamid Aghvami for his support, help and encouragement. Under his supervision, I learned a rigorous research attitude and a broad outlook on life. He has always emphasized the importance of fundamental knowledge in scientific research, which motivates me to study the basics patiently. He also has taught me to be confident and ambitious, which gives me power. And I am inspired to be diligent by his hard-working. Then, I want to give sincere thanks to my second supervisor Dr Vasilis Friderikos for his helpful remarks and advice on my research works.

Furthermore, I would like to thank my colleagues and friends in the Departments of Informatics and Engineering at King's College London for their continuous friendship, support and encouragement. I will always remember the joyful days we spent together during our PhD studies.

Special thanks to Dr Qiang Sheng for his constant love, care, help and support. Last but not least, I would like to express my deepest thanks to my dear parents for their unconditional love and continuous support. Undoubtedly, I could not have finished my PhD study without their support. They always give their best to support me in pursuing all I want. I would like to dedicate the thesis to my beloved mother and father.

Abstract

End-to-End network slicing is an emerging technology that provides substantial potential for supporting various services and carries flexible resource orchestration for 5G networks. The technology allows a physical infrastructure to be divided into several logical and virtual network slices based on the powerful Software-Defined Networking and Network Function Virtualization techniques. However, it is challenging to ensure high efficiency of resource utilization and energy consumption in the deployment of network slices by proposing design policies for different scenarios. Thus, it is necessary to investigate appropriate algorithms of design policy to facilitate the deployment of the network slices onto a shared underlying infrastructure in various scenarios. In the thesis, the proposal of design policy is based on multiple objectives. The primary design objectives are covered from three different aspects, containing the characteristics of diverse service scenarios, the fluctuated traffic demands in the network and the energy consumption of network slices. On this basis, the proposed algorithms aim to enhance resource and energy efficiency and satisfy various service requirements for different 5G use cases, including enhanced Mobile BroadBand (eMBB), massive Machine Type Communications (mMTC) and ultra Reliable & Low Latency Communications (uRLLC).

Firstly, a design problem of network slicing is proposed and formulated via a mathematical model. And a basic algorithm of design policy is presented, which can ensure various network slices in different use cases are deployed onto a shared underlying infrastructure with a certain number of resources. The proposed algorithm can deliver an efficient utilization of network resources and a balanced occupancy of the physical network. Nevertheless, the design policy fails to consider the uncertainty in service requirements. For instance, uncertainty can lead to fluctuating traffic demands.

Thus, in the following work, a service-aware design problem is formulated with the fluctuations in traffic demands, including a deterministic formulation and a robustness one. In addition, a heuristic algorithm is proposed to realize the mapping of network slices under fluctuation in resource requirements and guarantee the performance of different services. However, the design problem for network slices is an NP-hard problem, its exact formulation is incapable of obtaining optimal solutions within a reasonable computation time. Besides, it is very likely for heuristic algorithms to suffer from a locally optimal solution. Hence, thirdly, a Deep Reinforcement Learning (DRL)-based approach is introduced to investigate the design problem with an energy-aware objective. It is modeled as a Markov Decision Process problem with the elements of state, action and reward. Furthermore, a policy network is established based on the Pointer Network architecture. And the network model is trained by leveraging the Advantage Actor-Critic algorithm. It is found that the proposed design policy can guarantee an efficient energy cost. Finally, the thesis is summarized, and some promising research directions for the network slicing design problem are stated.

Table of contents

List of figures

List of tables

List of glossary & acronyms

Glossary

G	Physical infrastructure topology
P	Set of physical nodes
L	Set of physical links
$\mathbb{T}$	Set of relations of physical nodes and their attributes
A_P	Set of attributes of physical nodes P
A_L	Set of attributes of physical links L
Q	Network slice topology
$\mathcal{K}$	The number of network slice requests
V	Set of virtual network functions
E	Set of virtual links
p_i	A physical node
v_s	A virtual network function
q_k	A network slice request
$l_{i'j'}$	Physical link between the physical nodes $p_{i'}$ and $p_{j'}$ when they connect each other directly

$\mathcal{L}_{ij}$	Physical path between the physical nodes p_i and p_j when a forwarding node p_η exists between them, denoting as $\mathcal{L}_{ij} = \{l_{i\eta}, p_\eta, l_{\eta j}\}$
$h(\mathcal{L}_{ij})$	Latency of the physical path $\mathcal{L}_{ij}$
$b(\mathcal{L}_{ij})$	Bandwidth capacity of $\mathcal{L}_{ij}$
D_k	Set of traffic demands of q_k
$S(D_k)$	Set of sources of D_k
$H(D_k)$	Set of destination of D_k
e_{st}	Virtual link between the VNFs v_s and v_t
d_k	A traffic demand in D_k
c_{p_i}	CPU capacity of p_i
m_{p_i}	Memory capacity of p_i
I_{v_s}	Set of required attributes of v_s
c_{v_s}	Size of CPU resource blocks required by v_s
m_{v_s}	Size of memory resource blocks required by v_s
ε_{v_s}	Elastic coefficient of v_s
$c_{\lambda_{v_s}}$	CPU resource coefficient of v_s
$m_{\lambda_{v_s}}$	Memory resource coefficient of v_s
$\rho_i^{s,k}$	Decision variable of VNFs
$\gamma_{ij}^{st,k}$	Decision variable of virtual links
ζ_i	Variable of occupancy status of p_i
x_{v_s}	Number of resource blocks required by v_s
κ_{p_i}	Performing role of p_i

o_{p_i}	Occupancy state of p_i
a_δ	Energy consumption of per unit of CPU
z_δ	Energy consumption of per unit of bandwidth

Acronyms

NS	Network Slicing
NSDP	Network Slicing Design Problem
QoS	Quality of Service
NFV	Network Function Virtualization
SDN	Software Defined Networking
InP	Infrastructure Provider
VNF	Virtual Network Function
NSP	Network Service Provider
VN	Virtual Network
SP	Slice Provider
NO	Network Operator
SFC	Service Function Chain
NSI	Network Slice Instance
mMTC	massive Machine Type Communications
eMBB	enhanced Mobile Broadband
uRLLC	ultra Reliable & Low Latency Communications
VNE	Virtual Network Embedding

SFCP	Service Function Chaining Placement
CAPEX	Capital Expenditures
OPEX	Operational Expenditures
SLA	Service Level Agreement
SSI	Slice Service Instance
PSO	Particle Swarm Optimization
MOPSO	Multi-Objective Particle Swarm Optimization
MDP	Markov Decision Process
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
DQN	Deep Q-learning Network
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
LSTM	Long Short-Term Memory
MC	Monte Carlo
TD	Temporal Difference
DP	Dynamic Programming

DPG	Deterministic Policy Gradient
VM	Virtual Machine
VMM	Virtual Machine Monitor
NFVI	Network Function Virtualization Infrastructure
EMS	Element Management System
MANO	NFV Management and Orchestrator
VNFM	Virtual Network Function Manager
VIM	Virtual Infrastructure Manager
OSS	Operations Support System
BSS	Business Support System
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
NaaS	Network as a Service
NSaaS	Network Slicing as a service
RAN	Radio Access Network
CN	Core Network
TN	Transport Network
AMF	Access and Mobility Management Function
SMF	Session Management Function
UPF	User Plane Function

UDM	Unified Data Management
PCF	Policy Control Function
NRF	Network Function Repository Function
NSSF	Network Slice Selection Function
VA	Virtual Augmented
VR	Virtual Reality
V2X	Vehicle to Everything
LP	Linear Programming
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
NLP	Non-Linear Programming
SOP	Single-objective Optimization Problem
MOP	Multiple-Objective Optimization Problem
RO	Robust Optimization
SO	Stochastic Optimization
CO	Combinatorial Optimization
NCO	Neural Combinatorial Optimization
COP	Combinatorial Optimization Problem
MCDM	Multi-Criteria Decision Making
EMO	Evolutionary Multi-Objective Optimization
SAW	Simple Additive Weighting

TSP	Travelling Salesman Problem
FCA	Formal Concept Analysis
TTL	Time-To-Live

Chapter 1

Introduction

1.1 Motivation

Nowadays, an explosive growth of mobile data traffic and unpredictable connections among devices can stimulate tremendous pressure on the performance of traditional communication networks [1]. With the rapid development of wireless communication technologies, 5G networks and beyond systems are expected to support a wide range of services with various performance requirements for different application scenarios. Thus, Network Slicing (NS) technology has emerged as a powerful architecture in 5G networks, which allows the physical network to split into several virtual, isolated and logical End-to-End network slices [2]. Different network slices can be accommodated by a shared underlying infrastructure based on various requirements of Quality of Service (QoS).

Network slicing technology has been developed as an appropriate way for the network management and administration, which can provide multi-tenancy, scalability and flexibility. Specifically, it is built on several promising softwarization technologies, such as Network Function Virtualization (NFV) [3], Software-Defined Networking (SDN) [4], and Cloud computing [5]. NFV provides a new way to virtualize network functions and enables them to operate on proprietary hardware, promising the flexibility of resource management and orchestration. SDN is a developing technology that can decouple the data plane of the forwarding process of network packets from the control plane of the routing process. In addition, Cloud computing technology can allocate resources dynamically to run applications for remote end users through cloud architectures with a set of data center servers and software development platforms.

Network slices can be enabled by leveraging network virtualization technologies, which allow multiple virtual networks to be deployed onto the same physical infrastructure [6]. In the network virtualization, physical resources are managed and orchestrated by Infrastructure Providers (InPs). NFV technology can facilitate service operators to deploy Virtual Network Functions (VNFs) and Network Service Providers (NSPs) flexibly, which can integrate diverse network resources of multiple providers to generate customized Virtual Networks (VNs) [7]. Besides, network slices are provided by Slice Providers (SPs) for tenants to supply diverse services. To realize different services, tenants can apply for a new creation of network slices from Network Operators (NOs) based on their specific requirements. The services can be described by a set of Service Function Chains (SFCs) with a predefined sequence of VNFs [6].

Each use case can be served by a set of network slices. And every network slice consists of a set of Network Slice Instances (NSIs) including VNFs and abstraction of required infrastructure resources [4]. The network slicing technology can orchestrate and manage network resources flexibly, which can provide multiple services satisfying various performance requirements in different use cases.

Three fundamental types of 5G use cases have been identified: massive Machine Type Communications (mMTC), enhanced Mobile Broadband (eMBBs) and ultra Reliable & Low Latency Communications (uRLLC) [8]. Particularly, mMTC use cases can support massive device accesses sending small data packages. eMBB use cases express performance requirements on high traffic demands, and uRLLC use cases require millisecond latency and high reliability.

Although the network slicing technology brings substantial benefits to 5G networks, some intractable issues still need to be addressed. One of the vital problems is the Network Slicing Design Problem (NSDP), which is defined as how to efficiently guide the implementation of network slices constrained by limited infrastructure resources.

Technically, the NSDP is derived from the Virtual Network Embedding (VNE) technology [9, 10] and the Service Function Chaining Placement (SFCP) technique [11–13]. Despite this, the NSDP can not be solved by only leveraging these two techniques. The main reason is that the existing methods focused mainly on the reduction of Capital Expenditures (CAPEX) and Operational Expenditures (OPEX). They give little consideration on the inherent characteristics of network slices, especially for different service use cases.

To sum up, it is significant to study the network slicing design problem. And the problem should be extensively investigated in both academia and industries.

1.2 Network Slicing Design Problem Statement

The overall aim of the network slicing design problem is to deploy different network slices onto the physical infrastructure and efficiently allocate the required computational, storage and networking resources based on proper mechanisms while satisfying various Service Level Agreements (SLAs) of network slice tenants. The thesis aims to achieve the goal by proposing several algorithms of design policy with different design objectives constrained by limited resources of the underlying infrastructure, taking into consideration different characteristics of various use cases, fluctuated traffic demands in the network and the energy consumption of the network slices. The proposed design policies intend to satisfy the performance requirements of various network slices in different use cases and enhance the resource and energy efficiency of the network.

In the thesis, it is assumed that the VNFs and virtual links in different network slices are provisioned by the same underlying infrastructure. And the physical nodes and physical paths in the underlying infrastructure have the same priority to host every VNF and virtual link. End-to-End network slices are deployed to provide services in various 5G application scenarios for users. Each Slice Service Instance (SSI) is modeled as a set of VNFs and the required resources (e.g. computing, networking and storage resources), which can form a deployed network slice request to satisfy certain network characteristics. The required resources are allocated to the requests according

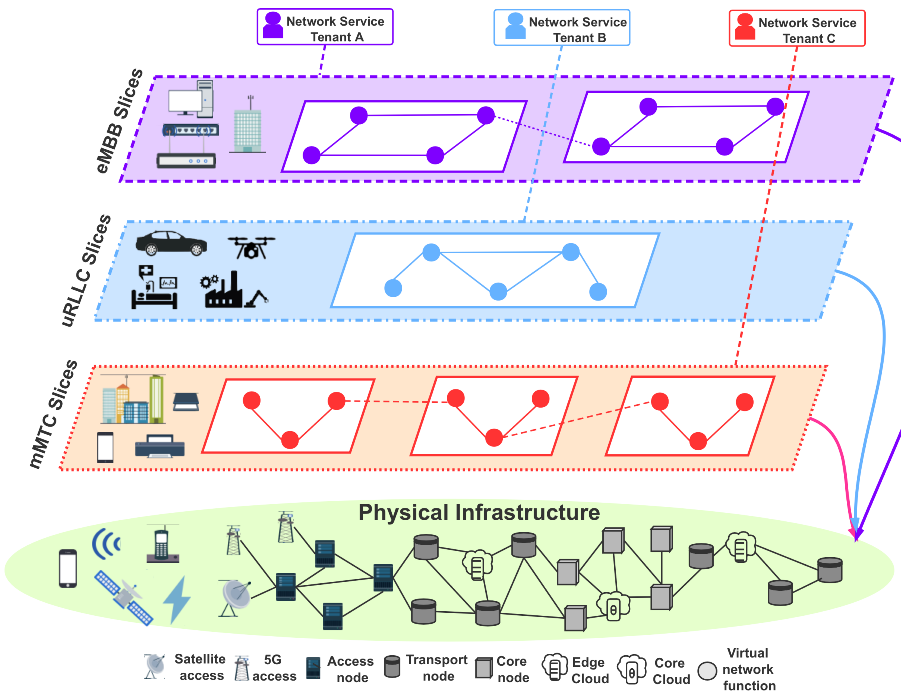


Fig. 1.1 Illustration of the network slicing design problem for various 5G use cases.

to a set of restrictions. Fig. 1.1 illustrates that different customized network slices including several SSIs are established onto the underlying infrastructure.

1.3 Research Contributions

The main contributions of the thesis are the proposal of the network slicing design problem with multiple design objectives and the delivery of the algorithms of design policies for network slices in 5G networks. The research contributions can be summarized as follows:

- The network slicing design problem has been presented and solved from multiple aspects with different design objectives. The proposal of design objective mainly focuses on the various network characteristics of eMBB, mMTC and uRLLC use cases, the fluctuation of traffic demands in the network and the energy cost of the network slices deployment.

  To be specific, a basic design objective is proposed for various use cases, on this basis, a service-aware design objective is presented with fluctuated traffic demands, both of them are considering the bandwidth resource utilization, the CPU and memory resource utilization and latency. Besides, an energy-aware design objective is given, which consists of the energy consumption of physical nodes and links.

- The formulation of an Integer Linear Programming (ILP) problem for the network slicing design problem is presented, which aims to satisfy the performance requirements in different 5G application scenarios and improve the resource and energy efficiency of the network.

  In particular, two optimization models of the service-aware network slicing design problem are formulated concerning uncertain traffic demands. A deterministic formulation is regarded as a nominal case with certain traffic demands. And a robust one is developed as an extended version of the deterministic formulation to deal with uncertain situations, where equivalent robust counterparts can be obtained with robust coefficients.

- A basic heuristic algorithm and a service-aware heuristic algorithm are proposed respectively to solve the network slicing design problem by obtaining sub-optimal solutions. Both algorithms are inspired by the Particle Swarm Optimization (PSO) algorithm. They are considered as a trade-off scheme between computational efforts and the quality of solutions.

  Specifically, in the basic algorithm of design policy, each particle is regarded as a design solution of VNFs in network slices. Furthermore, in the service-aware algorithm, each particle represents a final design solution of network slice, and it is assumed that each network slice contains a set of end-to-end traffic demands that may fluctuate after they steer through a collection of predetermined order VNFs. An initialization algorithm of particle swarms is presented to obtain a set of candidate solutions. An update scheme of particle positions is also proposed to guide the particles to approach the sub-optimal solution during exploration processes according to fitness values with fluctuated traffic demands.

- Regarding the energy-aware network slicing design problem, it is modeled as a Markov Decision Process (MDP) problem with a set of critical elements, including a finite state space, a finite action set, a transition probability and a reward function. The reward function is investigated in terms of the link energy consumption of network slices, which can combine the design process of VNFs with the design process of virtual links.

  Particularly, a Deep Reinforcement Learning-based algorithm, the Advantage Actor-Critic (A2C) algorithm, is employed to solve the energy-aware problem by learning design policies. The parameterized policy network as an actor network is optimized under the guidance of a critic network. The policy network is implemented based on the pointer network architecture with an attention mechanism, and both the actor and critic networks contain two Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) layers. And a search strategy is presented to refine and update the parameters of the policy network and verify the sampled solutions during the inference process for determining the final design solutions. The present algorithm architecture can provide flexibility and scalability in terms of the size of output sequences, which is appropriate to address the network slicing design problem with a variation of network slices.

Additionally, the work publications listed below present the research outputs of the thesis.

Publications:

- Ranyin Wang, A. Hamid Aghvami, Vasilis Friderikos. An End-to-End Network Slicing Design Policy. (2020). *In 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, pp. 1-6.

- Ranyin Wang, A. Hamid Aghvami, Vasilis Friderikos. Service-Aware Design Policy of End-to-End Network Slicing for 5G Use Cases. (2022). *IEEE Transactions on Network and Service Management*, 19(2), pp. 962-975.

- Ranyin Wang, A. Hamid Aghvami. Energy-aware Design Policy for Network Slicing using Deep Reinforcement Learning. (*Under review*)

1.4 Thesis Structure

The rest chapters of the thesis are organized as follows:

- Chapter 2 introduces the technical background of the research area in this thesis. The chapter firstly gives a brief introduction of key concepts in the End-to-End network slicing technology. Then, the fundamental knowledge of the network slicing technology is introduced, including the implementing principles and framework of network slices. A brief summary of 5G use cases is also presented. Besides, this chapter summarizes the enabling technologies of the network slicing design problem, including the Virtual Network Embedding, the Service Function Chain Placement, the Optimization Methods and the Deep Reinforcement Learning technology.

- Chapter 3 presents a basic network slicing design policy for different use cases. One of the vital aims of the network slicing design problem is to solve the deployment of different network slices and guarantee them coexist in the same physical infrastructure. Thus, in this chapter, an ILP formulation of the basic network slicing design problem is formulated with multiple design objectives for various use cases. And a heuristic algorithm of a versatile design policy is proposed to ensure different kinds of network slices can be deployed onto a shared physical network while satisfying the service requirements of various application scenarios.

- Chapter 4 proposes a service-aware network slicing design policy for different use cases concerning fluctuated traffic demands in the network. One of the key challenges is how to develop a proper deployment mechanism for mapping network slices onto the physical network and allocating their required resources for satisfying the requirements of various network services, especially when the requirements are uncertain due to traffic fluctuation. Therefore, in this chapter, two optimization models, the deterministic formulation and the robust formulation, are proposed to deal with the uncertain situations in the design processes of network slices. Moreover, a heuristic algorithm is presented to deploy network slices, which aims to utilize network resources efficiently while ensuring different service performances of network slices.

- Chapter 5 investigates an energy-aware network slicing design policy taking into consideration of the energy consumption of the network. One of the critical principles for solving the energy-aware network slicing design problem is to ensure a maximum number of network slices can be accommodated by the underlying infrastructure while saving energy in the deployment of network slices. Hence, in this chapter, an energy-aware objective for the network slicing design problem is proposed, and the energy consumption of the network is investigated from two aspects, the energy costs of physical nodes and physical links. Besides, the energy-aware problem is modeled as a Markov Decision Process problem with Reinforcement Learning elements. And a DRL-based algorithm, the Advantage Actor-Critic algorithm, is leveraged to solve the problem with the help of the pointer network architecture and policy gradient mechanism.

- Finally, Chapter 6 concludes the thesis. Future insights and potential directions in the research of network slicing are also stated.

Chapter 2

Background

2.1 Fundamental Concepts for End-to-End Network Slicing

5G networks are expected to support a great variety of vertical industries that may require different service performances. Mobile network operators aim to provide customers with a 5G service of various characteristics, such as End-to-End, flexibility, scalability, demand-aware and security. End-to-End Network Slicing is a crucial technology to achieve the goal where network slices can be customized to satisfy different service requirements of various application scenarios. The realization of network slices mainly depends on a range of critical techniques, such as Virtualization, Containerization, Softwarization, Virtual Machines, Containers, SDN, NFV, Cloud Computing and Isolation, which enable physical resources to be shared among various network slices effectively. In this section, the fundamental concepts of the End-to-End network slicing are introduced as follows.

2.1.1 Virtualization

Virtualization is a powerful technology that can simplify the development and testing of systems, which refers to the creation of virtual resources, such as servers, operating systems and networks [14]. It is an important process for network slicing as it can enable flexible and dynamic network orchestration and management to tackle the network ossification problems by allowing various virtual networks to share the same resource pool of a physical network.

The main goal of virtualization is to create a virtual version of the physical hardware for managing resources and workloads flexibly. Virtualization can emulate hardware devices by using software applications, which makes traditional dedicated resources more scalable and programmable. Virtualization provides many benefits, such as low or no-cost deployment, sufficient resource utilization, operational cost savings and power savings [15]. Besides, the virtualization can be applied to different system layers, containing the CPU virtualization, memory virtualization and device/IO virtualization. Each kind of virtualization technology has its own set of strengths and complexities.

2.1.2 Virtual Machines & Containers

Each Virtual Machine (VM) hosted by the same physical device shares the same hardware resources, such as computing resources, storage and memory resources and networking resources, while they are isolated from each other and the host [16]. Containers are created based on the operating system-level virtualization technology, which are light-weight hypervisor-based VMs. The differences of the structures of VMs and containers are shown in Fig. 2.1.

Containerization is a kind of virtualization technology that has been developed as an alternative to the conventional hypervisor-based virtualization strategy [17]. In containerization, different containers share the same virtualized physical server rather than create it for each VM. Docker is a typical example of container virtualization platform [18].

Furthermore, the layer between the physical hardware and the operating system is responsible for creating, controlling, supervising and orchestrating guest VMs, and it is defined as Virtual Machine Monitor (VMM) or hypervisor. Hypervisor can supervise the sharing of physical resources among heterogeneous virtual networks. Two main types of hypervisor have been investigated, namely the type-1 (bare metal hypervisor),

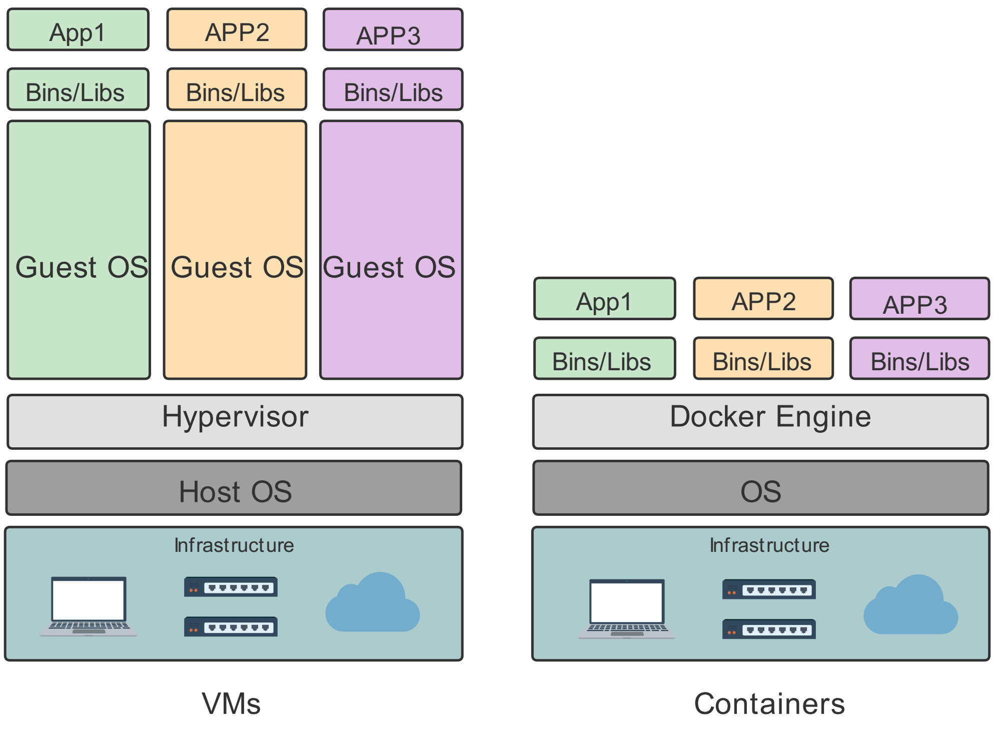


Fig. 2.1 Structures of VMs and containers.

i.e., XEN [19], VMware [20], KVM [21], and the type-2 (hosted hypervisor), like Oracle Virtual Box, similarly to other computer applications.

2.1.3 Network Function Virtualization

Network Function Virtualization (NFV) provides a promising approach to design, deploy and manage networking services, which is developed with the help of the evolution of IT virtualization [7]. NFV can separate Network Functions (NFs) from the physical hardware devices. It also enables to transfer NFs from underlying dedicated devices to software-based applications running on commercial servers.

Besides, NFV can bring many benefits to the telecommunications industry, such as the openness of platforms, scalability and flexibility and the reduction of CAPEX and OPEX investments [3]. It aims to assign Virtual Network Functions (VNFs) to the hardware facilities flexibly and enhance the network operations effectively. To achieve the goal promised by NFV, the ETSI NFV group [22] has standardized the

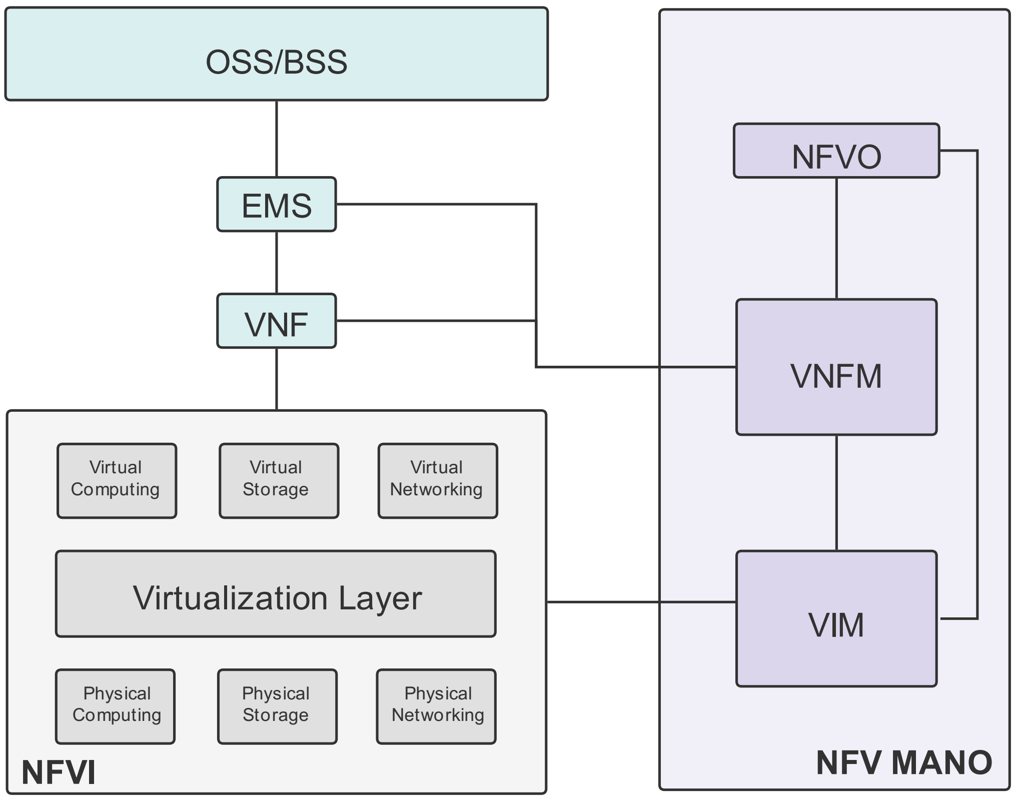


Fig. 2.2 Framework of the NFV architecture.

architectural framework of NFV by decoupling NFs from the proprietary hardware appliances, which ensures the NFs can be performed in software.

The architectural framework of NFV is illustrated in Fig. 2.2, which consists of several functional components, such as VNFs, Network Function Virtualization Infrastructure (NFVI), Element Management System (EMS), NFV Management and Orchestrator (MANO), VNF Manager (VNFM), Virtual Infrastructure Manager (VIM), Operations Support System (OSS) and Business Support System (BSS) [23]. Specifically, the NFVI is required to run VNFs that are managed by the EMSs. MANO manages the lifecycle of network services consisting of a set of VNFs, VNFM manages the lifecycle of VNFs, and VIM supervises and orchestrates the NFVI associated resources.

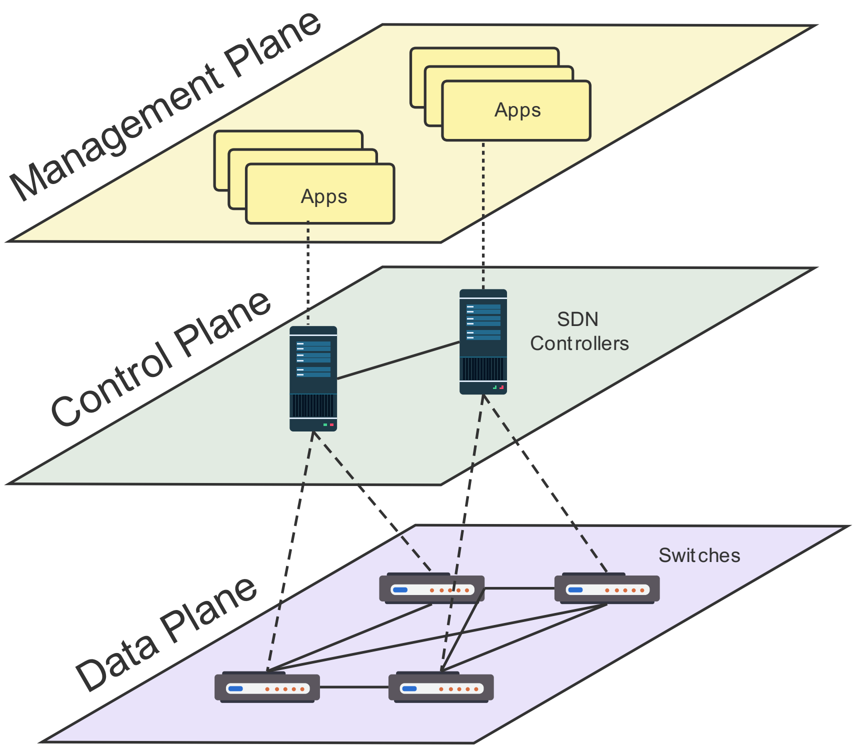


Fig. 2.3 Framework of the SDN architecture.

2.1.4 Software Defined Networking

Software Defined Networking (SDN) is a complementary methodology to NFV for network management. SDN refers to a network architecture where the forwarding status in the data plane is managed by the remote control plane decoupled from the forwarding plane. The architecture is appropriate to support 5G network slicing and to provide important characteristics that are necessary for implementing network slicing, such as programability, scalability and service-oriented adaptation [24]. The illustration of the architecture of SDN is shown in Fig. 2.3.

Additionally, the main SDN architectural components are logically centralized controllers, which can manage network slices effectively and dynamically based on the key principles of network slicing. SDN can provide a centralized view of networks, which enables the SDN controllers to act as a orchestrator in networks [25]. SDN leverages centralized and programmable technology to enhance the possibility of adjusting to the rapidly changing needs of industry and business. Besides, SDN can promise lower costs and reduce wasteful provisioning, and it can also provide flexibility and innovation for networks.

2.1.5 Cloud & Edge Computing

Cloud computing can provide storage, computing and networking resources in a single or several remote public platforms to enable a network slice [1]. Particularly, it allows physical servers to host one or more VMs on demand, which offers efficient resource allocation of servers by using cloud architectures. Further, Cloud computing can allocate resources and carry out applications for remote end users dynamically by leveraging the data center servers and software development platforms. There are three typical categories of Cloud computing services: private, public and hybrid. Moreover, the services can be categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [5]. In recent days, cloud deployments have a rapid development with efficient cost savings and greater flexibility over the conventional private data centers.

Edge computing is a promising technology in 5G, which is expected to provide low-latency communications. Edge computing can move the system resources, such as computing, storage, and networking resources, from remote public clouds to the edge of networks [26]. Hence, mobile customers can experience a low end-to-end latency by requiring virtual resources from the access network.

2.1.6 Isolation

Isolation is a basic property for network slicing, the cooperation of virtualization and orchestration can ensure the required isolation levels of each network slice are realized [4]. The isolation of network slices aims to guarantee the service-based performance requirements and the security and privacy of tenants [4]. To be specific, the isolation performance must be assured as multiple slices are performed on a shared infrastructure. Without proper isolation among different network slices, attacks and faults may be launched continuously. Attackers can attack network slices from one to others. Thus, network slices must maintain proper isolation and security functions to prevent unauthorized accesses.

To achieve this goal, several measurements shall be considered. For instance, the physical resources allocated to various network slices should be different. Each network slice can be regarded as a set of resources configured on the infrastructure, and the allocated resources must have no impact on others. However, it may result in inefficient utilization of network resources since the extra cost is needed for the explicit use of separating resources. Besides, potential attacks in a single network slice should be avoided by developing an appropriate strategy, which can exclude illegitimate accesses.

2.2 End-to-End Network Slicing Technology

An End-to-End network slice is logically managed as a virtual network, which may contain the capabilities of Radio Access Network (RAN), Core Network (CN) and Transport Network (TN). It aims to provide a Network as a Service (NaaS) for various use cases, which allows the mobile network operators to create multiple virtual networks on a shared physical infrastructure. In 5G networks, three fundamental use cases of network slicing have been identified, such as eMBB, uRLLC and mMTC. The details of the three application scenarios are introduced in the next Section 2.3.

End-to-End network slicing is the fundamental technology in 5G networks, which can provide better network performance than the traditional one-size-fits-all network architecture. Network slices can be customized based on different service requirements, where the allocation and utilization of physical resources can be optimized. Besides, each network slice contains its own network topology, VNFs, virtual resources and traffic flows.

In particular, RAN network slicing can be realized through a logical abstraction of physical radio resources and hardware, such as spectrum resources and base stations [26]. RAN slicing can be implemented by employing the Cloud RAN (C-RAN) architecture. C-RAN is a cloud computing-based architecture for RAN, which leverages the virtualization technique enhanced in cloud computing to dynamically share the physical resources and provide support to multiple end users. Besides, core network slicing is implemented to fulfill diverse service requirements by multiple VNFs, such as Access and Mobility Management Function (AMF), Session Management Function (SMF), User Plane Function (UPF), Unified Data Management (UDM), Policy Control Function (PCF), NF Repository Function (NRF) and Network Slice Selection Function (NSSF) [1]. Core network slicing aims to achieve dynamic management and orchestration of network resources. Specifically, mobile operators can design, create, manage, modify and delete the dedicated network slices for specific application scenarios dynamically. Regarding the transport network slicing, large bandwidth and low latency are necessary to satisfy the requirements of different use cases. The same transport network is expected to support different application scenarios.

In addition, the principles and framework of network slicing are introduced as follows.

2.2.1 Network Slicing Principles

Basic principles of implementing network slices are listed as follows [27]:

- High-Reliability and Isolation: Security of network slicing needs to be guaranteed first. High-reliable and isolated services should be provided to tenants who requested various performance requirements.
- Programmability and Scalability: Flexibility of network slicing is a key characteristic that is different from the conventional one-size-fits-all network system. Network slices should be updated and programmed to fulfill new service requirements flexibly.
- Dynamic and Automation: Automatic management of network slicing is necessary. The creation, deployment, deletion and update of network slices should be orchestrated automatically. Especially, network resources in network slicing should be utilized efficiently. The elasticity of network resources can be realized through a dynamic allocation scheme where the resources are allocated on a dynamic scale.

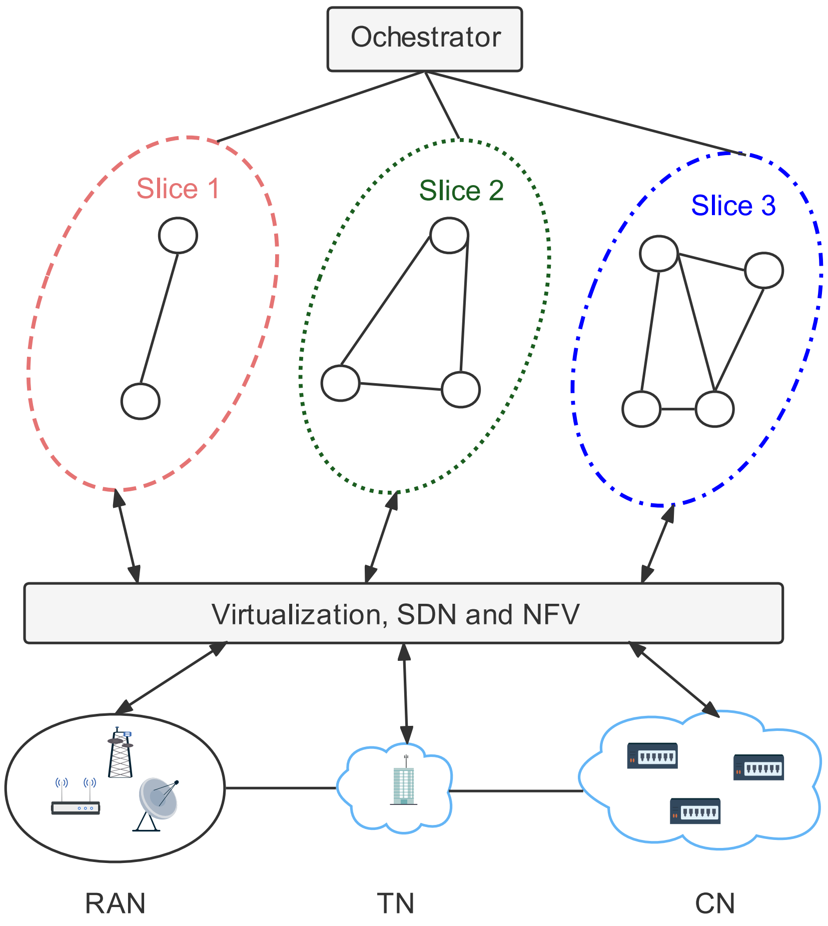


Fig. 2.4 The overall network slicing framework.

- Customization: Network slices are customized to provide various services based on different requirements of tenants. The customization of network slices can be achieved not only by leveraging the SDN technology that can decouple the data and control planes but also by using NFV technology that can enable service-oriented VNFs.

2.2.2 Network Slicing Framework

The overall framework of network slicing consists of four fundamental layers, including the infrastructure layer, the virtualization layer, the network slicing layer and the network slicing management layer, which is illustrated in Fig. 2.4.

To be specific, the infrastructure layer is formed of basic network capabilities of RAN, TN and CN, which can provide physical resources to support network slices,

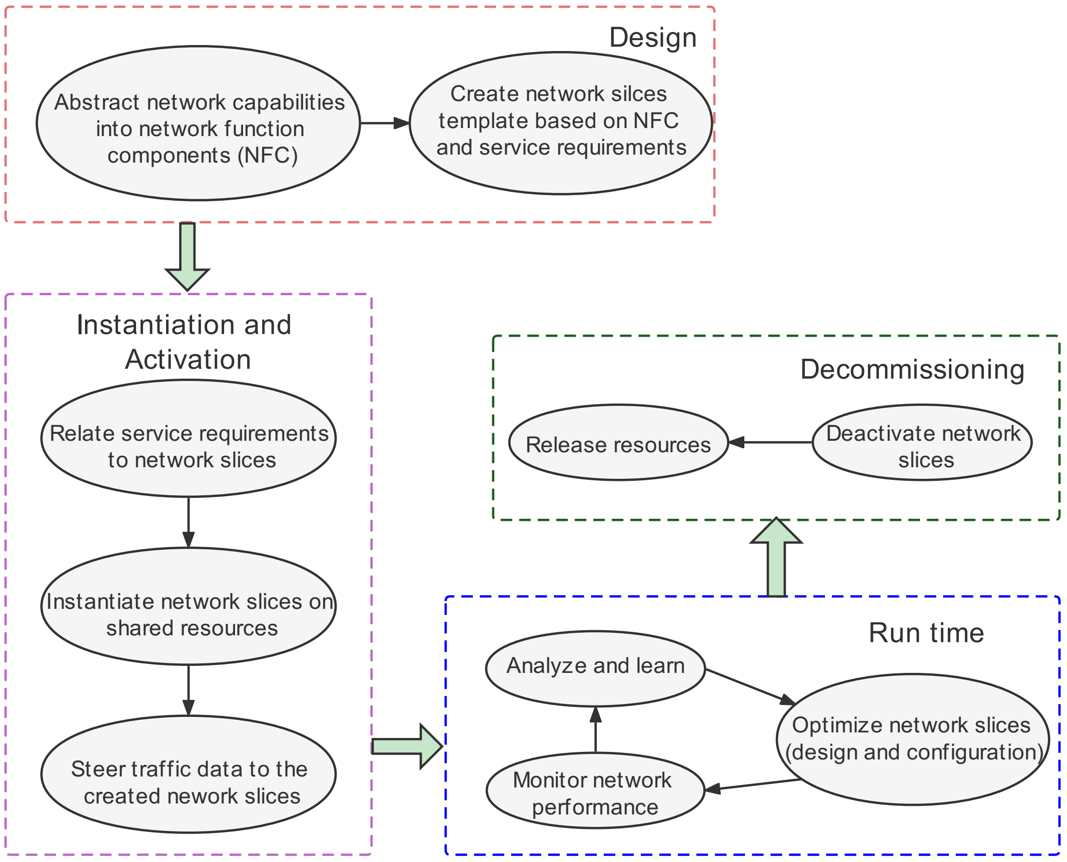

Design
Abstract network capabilities into network function components (NFC)
Create network silces template based on NFC and service requirements
Instantiation and Activation
Relate service requirements to network slices
Instantiate network slices on shared resources
Steer traffic data to the created nework slices
Decommissioning
Release resources
Deactivate network slices
Run time
Analyze and learn
Optimize network slices (design and configuration)
Monitor network performance

Fig. 2.5 Life cycle management of network slices.

such as storage resources, computing resources and networking connectivity. In the virtualization layer, virtual resources are abstracted in terms of attributes. NFV makes the VNFs independent of the underlying hardware devices, and SDN enables the creation of different isolated network slices that are completely decoupled from the infrastructure. The network slicing layer can safely run on top of the infrastructure through the virtualization layer. Service-oriented VNFs are deployed to form End-to-End network slices based on the network slicing instances.

Moreover, the network slicing management layer is introduced to enable flexible and automatic management of network slices. It is a critical part of the network slicing framework, which designs and manages network slices based on the scope of NFV MANO framework. The network slicing management layer consists of the OSS/BSS functions and the network slice MANO system [28]. Specifically, the network slice

Table 2.1 Differences among eMBB, uRLLC and mMTC use cases.

Characteristic	mMTC	uRLLC	eMBB
Availability	Regular	Very High	Regular (baseline)
E2E latency	Not highly sensitive	Extremely sensitive	Not highly sensitive
Throughput type	Low	Low/med/high	Medium
Density	High	Medium	High
Network coverage	Full	Localized	Full

MANO is the development of the NFV MANO, which contains the network slice templates management and the life cycle management of network slices [26].

The network slice MANO aims to create and manage VMs by utilizing system resources. The resources are allocated to the VNFs so that they can be connected to form different service function chains. Furthermore, the network slice MANO can manage the life cycle of network slices by interacting with the network slicing layer. The life cycle of network slices consists of four logical phases [2], and the details are illustrated in Fig. 2.5.

2.3 5G Use Cases

Each 5G use case, such as eMBB, uRLLC and mMTC, can be served by a set of network slices to provide its corresponding service scenarios, for instance, Ultra-HD Videos, Virtual/Augmented Reality (VR/AR), Automotive Network and Vehicle to Everything (V2X), Smart Grids, Remote Medical Services including Remote Surgery, Motion Controls and Intelligent Cities. Specifically, different performance requirements, such as low latency, i.e., 1-10ms, and high reliability, enhanced data rate of 10Gbps and connection density with 10^6 devices per km^2 are required in 5G networks [1]. The summary of the differences among eMBB, uRLLC and mMTC use cases is illustrated in Table 2.1.

2.3.1 Enhanced Mobile Broadband Communications

eMBB use cases aim to provide broadband accesses with up to 10Gbps bandwidth to enable different service scenarios, such as dense urban society, i.e. stadium, Ultra-High Definition (UHD) Videos streaming, Cloud Storage, Moving Hot-spots and AR. They facilitate the support for the services with high data rates. Specifically, they can deal with huge data traffic volumes and offer wide area connectivity and coverage. Moreover, they can support the scenarios with high user mobility and enhanced broadband, for instance, the extremely fast moving vehicles, including high speed trains and drones. For most eMBB use cases, the latency should be low, i.e., 5-10 ms, and the minimum availability and reliability should be 95% respectively [29].

2.3.2 Ultra Reliable & Low Latency Communications

uRLLC use cases can assure the services with ultra-low latency connectivity, ultra-high reliability and availability, such as interactive tactile Internet, automated traffic control, automatic driving, AR/VR, collaborative robots and remote object manipulation. The services require a very low latency, i.e., 1ms and a very high reliability and availability with minimum 99.9% and 99.999% respectively. Besides, the service scenarios of e-healthcare, lifeline communications and public safety, such as disaster relief and emergency response, require a low latency, i.e., 5ms, and a very high reliability and availability with minimum 99.999% and 99.999%, respectively, with packet loss as low as 1 packet out of every 10^4 packets [29].

2.3.3 Massive Machine Type Communications

mMTC use cases can facilitate the network connectivity for high density of devices, especially the non-latency sensitive devices, in ultra-dense scenarios with broadband accesses. They can guarantee the high density of network connectivity for users with various smart and intelligent devices, i.e., Smart-phones and Smart Wearables, in the area of Smart Homes/Cities and Smart Farming. The use cases can relatively be tolerant to the latency, for instance, 10ms. And they require a normal to high reliability and availability with minimum 95% respectively [29].

2.4 Enabling Technologies of Network Slicing Design Problem

Network slicing is a key component of the 5G networks. A significant challenge in the research of network slicing is how to guide a practical deployment and implementation of network slices, which spawns the NSDP. In this section, the basics of enabling technologies for addressing the NSDP are introduced.

Specifically, two fundamental enabling technologies: VNE and SFCP are firstly illustrated. Then, the optimization methods are introduced, including Integer Linear Programming, Multi-Objective Particle Swarm Optimization, Robust Optimization, and Neural Combinatorial Optimization. Besides, the details of DRL approaches are given, containing the general idea behind DRL, the fundamental definitions, the value-based DRL methods and the policy-based DRL approaches.

2.4.1 Virtual Network Embedding

One of the most difficult challenges faced in the practical deployment process of network slices is to determine a feasible design policy. The slicing design process is mathematically known as a VNE problem. Thus, the NSDP with the goal of deploying different network slices onto the physical network can be regarded as a VNE-type problem. To be specific, embedding a set of virtual networks to a physical infrastructure is proven to be a NP-Hard problem [30], which means that the problem is challenging to be solved within a polynomial computing time even for small to medium network instances. As a result, the studies about VNE have received extensive attentions from researchers.

An example of a VNE problem is illustrated in Fig. 2.6. Specifically, in the figure, regarding the physical nodes $A-F$, the numbers (2,16), (4,8), (4,32), (8,16), (2,8) and (8,8) next to $A-F$ represent their physical resources capabilities, such as CPU

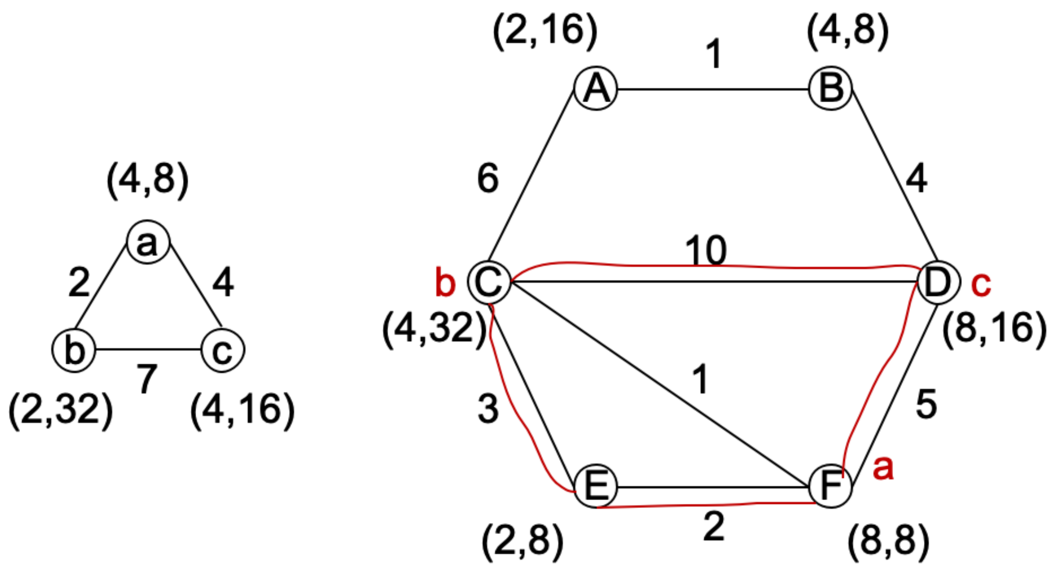


Fig. 2.6 Illustration of an exampled VNE problem.

and storage. And the numbers next to the physical links among the physical nodes $A-F$ indicate their bandwidth capabilities. Similarly, the numbers next to the VNFs $a-c$ and their virtual links indicate their resource requirements on demand. In the given example, obviously, the VNFs a, b and c can be mapped onto the physical nodes F, C and D, respectively, and the virtual links $a \to b$ and $b \to c$ can be placed onto a physical path composed of the physical links $F \to E$, $E \to C$ and $C \to D$.

Many studies on the VNE have been devoted to exploring heuristic algorithms to embed virtual networks onto the same infrastructure. For instance, Yu *et al.* [30] reconsidered the VNE problem and proposed an algorithm to consider the flexible path splitting and migration, which contained two phases: the virtual nodes mapping and the virtual links mapping. Later, Chowdhury *et al.* [31] developed two classic VNE algorithms, D-ViNE and R-ViNE, presenting an enhanced correlation between two phases: the node mapping phase and the link mapping phase, via deterministic and randomized rounding techniques. And a fast-convergent heuristic algorithm was proposed in Reference [32] based on the Particle Swarm Optimization algorithm by considering the network topology, which aimed to provide a solution to embed a sequence of virtual networks into the underlying infrastructure.

Additionally, the Deep Reinforcement Learning methods introduced in Subsection 2.4.4 have also been applied to solve the VNE problem, including the policy gradient mechanisms and the Deep-Q learning methods. For instance, Yao *et al.* [33] proposed a continuous-decision VNE problem based on a Reinforcement Learning method, which was intended to solve the problem that the continuity of nodes embedding in virtual network requests was ignored. The authors modeled a continuous process of the node embedding by RNNs and updated network parameters through a basic policy-gradient approach. Reference [34] aimed to solve an automatic VNE problem by combining a Deep Reinforcement Learning method with a neural network structure of the Graph Convolutional Network. Specifically, the Asynchronous Advantage Actor Critic (A3C) algorithm was adopted to train the policy generation algorithm, which improved the efficiency of training procedures.

Moreover, in Reference [35], a DRL-based algorithm was proposed to solve the VNE problem. However, the representation of physical infrastructures and the changes of underlying network resources were both ignored during training processes. Besides, Dolati *et al.* [36] developed a DQN-based VNE problem named DeepViNE, which could automate the collection of problem features required in DRL methods. The key idea of this work was to encode physical and virtual networks as two-dimensional images by using a deep convolutional neural network. And a long-term reward of the proposed algorithm was formulated to minimize the failures of networks embedding. Solozabal *et al.* [37] employed a Reinforcement Learning approach to formulate a VNF placement policy by extending the neural combinatorial optimization theory. In this work, a neural network model was proposed to solve the placement problem by minimizing the overall energy consumption.

Despite the extensive attention on the VNE problems, the existing approaches and works do not take into account the heterogeneous application scenarios and different domains in 5G networks. The embedding problems will become harder and more complex when different use cases and service scenarios are considered. In tradition, the VNE approaches are proposed to map the virtual networks with only one type and to allocate the expected resources based on a single objective, which is not the case in

5G network slicing. Obviously, it is infeasible to solve the NSDP in 5G networks by adopting the existing VNE works directly. Thus, the new technologies for deploying network slices in 5G are significant to be researched by the scientific community.

2.4.2 Service Function Chain Placement

In general, a Service Function Chain (SFC) is an ordered and connected chain of VNFs for processing traffic flows with QoS requirements, which aims to deliver network services in virtual networks [38]. SFC placement mechanisms can support network services with the capabilities of the NFV and SDN technologies, which can define and instantiate an ordered set of VNFs and the flowing traffic through these functions [13].

The SFC placement mechanisms can automatically set up virtual network connections to handle different types of traffic demands via calculating an optimal routing path. Network services can be provided when traffic flows steer through a SFC according to performance requirements. The mechanisms can also address the challenges of the dynamic formation of service function chains for traffic flows. Thus, it is a significant enabling technology for solving the NSDP.

An example of a SFC placed onto a physical infrastructure is illustrated in Fig. 2.7. To be specific, a service function chain $VNF1 \rightarrow VNF2$ with the source node *src* and destination node *dst* can be placed onto the physical infrastructure and routed through an optimal physical path for delivering a service.

Traditional SFC placement problems have been studied extensively. Specifically, in Reference [39], a SFC placement problem in the MEC-NFV scenario was formulated using weighted graphs, and a Linear Programming-based approach and a heuristic algorithm were presented to solve the problem by maximizing resource utilization. Jang *et al.* [40] succeeded in maximizing acceptable flow rates and minimizing the energy cost by formulating an optimization problem of dynamic SFC placement. Reference [41] formulated a multiple objective optimization model to optimize the deployment cost by mapping VNFs and virtual links in SFCs. Abdelaal *et al.* introduced a novel approach for the SFC placement problem to achieve a load balancing over the core links and an efficient utilization of energy and bandwidth based on multiple resource constraints

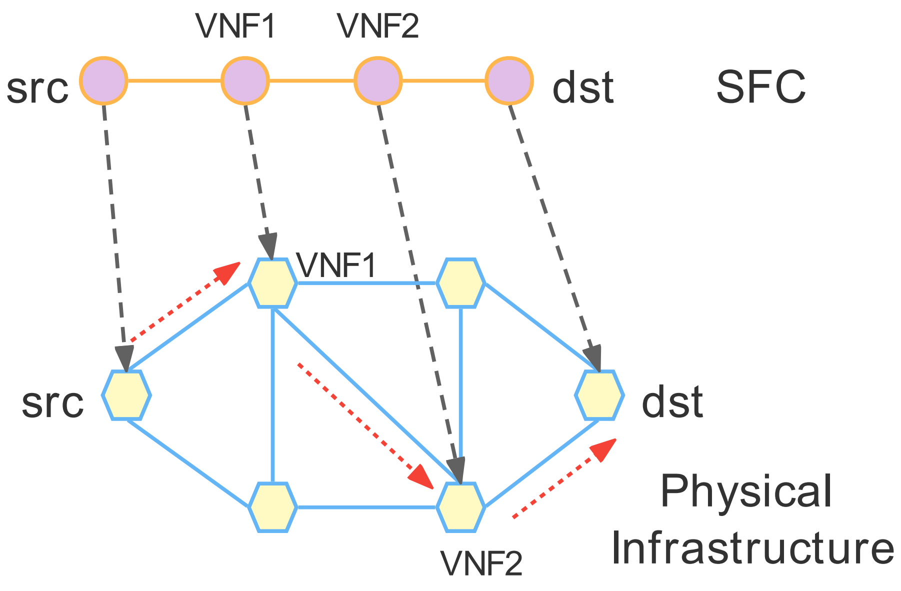


Fig. 2.7 An example of SFC placement on physical infrastructure.

[42]. Reference [43] focused on the multiple routing of data flows and intended to place SFCs in virtual networks. The work formulated the VNF placement and routing problem as a MILP problem to minimize the accumulated delay by considering the overall delay cost and the routing cost.

However, the emerging 5G brings new challenges to the existing SFC placement problems. Specifically, the SFC placement methods applied to solve the NSDP must consider diverse slice service requirements with different QoS parameters for various scenarios in 5G, which will increase the complexity of the problem compared to only taking into account sole scenarios. Besides, each network slice in various use cases requires distinct network behaviours, and different priorities of network characteristics should be considered in the SFC placement processes. Thus, it is crucial to investigate new approaches to solve the NSDP.

2.4.3 Optimization Methods

In this subsection, the optimization methods used to solve the NSDP are introduced. Firstly, a basic optimization problem can be formulated as:

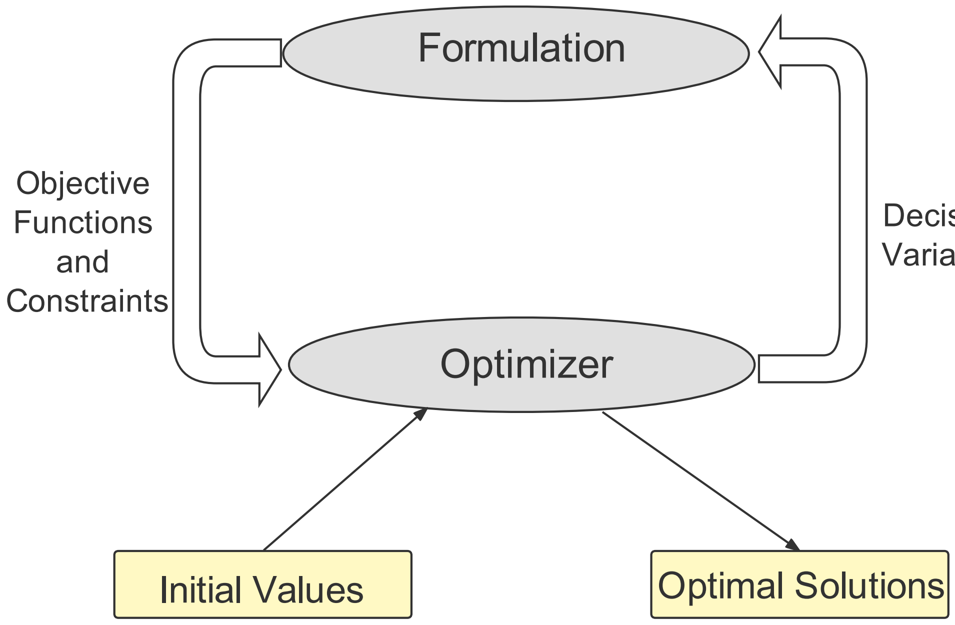


Fig. 2.8 Framework of a basic optimization problem.

$$\text{Optimize} \quad Y = y(x), \tag{2.1}$$

subject to

$$\begin{aligned} f(x) &= 0, \\ g(x) &\leq 0. \end{aligned} \tag{2.2}$$

The goal of an optimization problem is to decide a decision variables x that can optimize the objective function Y while guaranteeing that the formulation can produce feasible solutions limited by the equality constraints f and the inequality constraints g. The framework of a basic optimization problem is illustrated in Fig.2.8.

Among the ongoing researches of network slicing, in particular, regarding the issues about the deployment strategies and resource allocation mechanisms of network slices, the optimization methods have been explored extensively.

For instance, Reference [44] presented an inter-domain resource allocation scheme for network slices to maximize social welfare among tenants while minimizing operational expenditures for InPs. Fossati *et al.* [45] proposed an optimization framework for a fairly sharing multiple resources between slices. And a resource allocation scheme was formulated in Reference [46] as a convex optimization problem, where a distributed solution was introduced for the resource allocation problem between slices and data centers. In addition, Tajiki *et al.* [47] developed a novel architecture of resource allocation optimization to jointly manage the VNFs placement and routing, aiming to reduce the energy consumption.

Specifically, the main optimization methods applied in the research of network slicing can be summarized in the following categories [48] according to different types of decision variables, objective functions and constraints.

- Linear Programming (LP): the objective function and constraints are both linear. The decision variables can be integer or continuous. The LP problems consist of the Integer Linear Programming (ILP) problems and Mixed Integer Linear Programming (MILP) problems.
- Nonlinear Programming (NLP): the objective function or/and constraints are nonlinear. The decision variables can be scalar or continuous. The Mixed Integer Nonlinear Programming (MINLP) problems contain integer and continuous decision variables.
- Multiple-Objective Optimization Problems (MOP): there are more than one objectives in these problems.
- Robust Optimization or Stochastic Optimization (RO or SO): the objective function or/and the constraints have uncertain variables.
- Neural Combinatorial Optimization (NCO): a scheme to solve the Combinatorial Optimization Problems (COP) by using the Reinforcement Learning methods and Neural Networks.

In the following, the fundamental knowledge of the optimization methods used in the thesis are introduced.

Integer Linear Programming

The ILP problems involve a linear objective function and linear constraints with integer decision variables. The generalized formulation of an ILP problem can be stated as:

$$\text{Optimize} \quad Y = \sum_{i=1}^{N} A_i x_i, \tag{2.3}$$

subject to

$$\begin{aligned} &\sum_{i=1}^{N} a_{ji} x_i \leq b_j, \\ &j = 1, 2, ..., m, \\ &x_i \in \mathbb{Z}. \end{aligned} \tag{2.4}$$

If all of the decision variables x_i $(i = 1, 2, ..., n)$ are restricted on binary values $\{0, 1\}$, then the ILP problem is considered as a binary optimization problem. It is also a special case of the Discrete Optimization Problems.

Besides, it is clear to compare any given pair of solutions to decide on a better one when an ILP optimization problem has a single-objective. Consequently, a single-objective optimization problem is usually formulated.

Multiple-Objective Optimization

The optimization problems are called Multi-objective Optimization Problems (MOPs) if they have multiple objectives. Generally, it is difficult to determine a straightforward approach to obtain an optimal solution for the MOPs. Because the different objectives are usually defined in an incomparable way, specifically, one objective cannot gain a performance enhancement without deterioration of at least another objective. Thus, instead of deciding an optimal solution, the Pareto optimal solutions or non-dominated solutions are commonly obtained to determine a set of alternatives for the trade-offs of different objectives.

However, in practice, a preferred solution should be selected by a decision maker for implementation. To be specific, there are two main methods are employed to solve

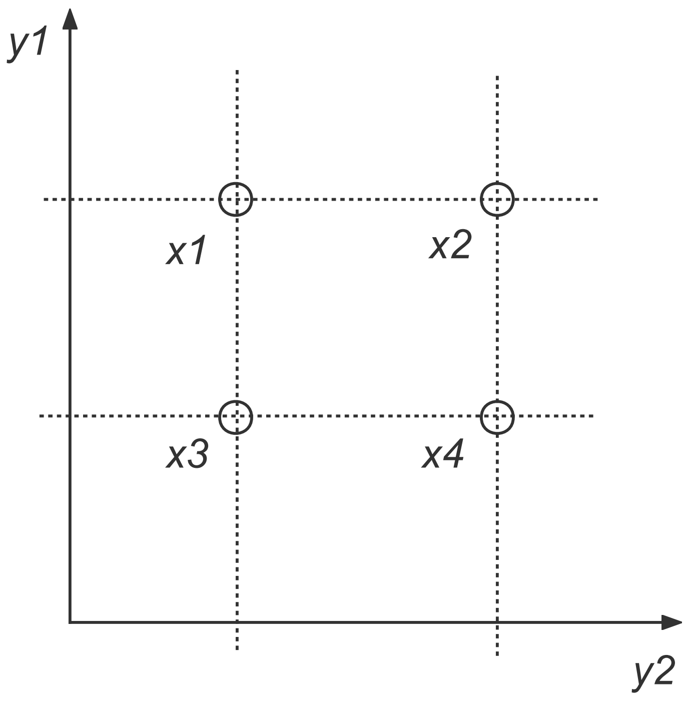


Fig. 2.9 Illustration of the concept of the Pareto dominance relation.

the MOPs, including the Multi-Criteria Decision Making (MCDM) approach [49] and the Evolutionary Multi-Objective Optimization (EMO) method [50].

Formally, a generalized MOP can be defined as:

$$\text{Optimize} \quad Y = \{y_1(\mathbf{x}), y_2(\mathbf{x}), ..., y_k(\mathbf{x})\}, \tag{2.5}$$

subject to

$$\begin{aligned} f(\mathbf{x}) &= 0, \\ g(\mathbf{x}) &\leq 0. \end{aligned} \tag{2.6}$$

where the decision vector **x** consists of a set of decision variables.

In the MOPs, the Pareto dominance relation [51] is usually adopted to obtain the optimal solutions. Assume $\mathbf{x}_a$ and $\mathbf{x}_b$ are two decision vectors for minimizing (2.5), $\mathbf{x}_a$ can Pareto dominate $\mathbf{x}_b$, expressing as $\mathbf{x}_a \prec \mathbf{x}_b$, if $Y(\mathbf{x}_a) \leq Y(\mathbf{x}_b)$ $(\forall i = \{1,2,...\})$ and $Y(\mathbf{x}_a) < Y(\mathbf{x}_b)$ $(\exists i = \{1,2,...\})$.

Fig. 2.9 illustrates an example of the Pareto dominance relation in a minimized problem. Specifically, $\mathbf{x}_3$ can Pareto dominate $\mathbf{x}_2$, noting as $\mathbf{x}_3 \prec \mathbf{x}_2$, because $y(x_i^3) \leq y(x_i^2)$ for $\forall i = \{1,2,...\}$ and $y(x_i^3) < y(x_i^2)$ for $\exists i = \{1,2,...\}$. Besides, $\mathbf{x}_3$ can also Pareto dominate $\mathbf{x}_1$ since $y_1(\mathbf{x}_3) < y_1(\mathbf{x}_1)$ even though $y_2(\mathbf{x}_3) = y_2(\mathbf{x}_1)$. Similarly, $\mathbf{x}_3 \prec \mathbf{x}_4$, $\mathbf{x}_4 \prec \mathbf{x}_2$ and $\mathbf{x}_1 \prec \mathbf{x}_2$. However, $\mathbf{x}_1$ and $\mathbf{x}_4$ can not dominate each other, which can be represented as $\mathbf{x}_1 \nprec \mathbf{x}_4$ and $\mathbf{x}_4 \nprec \mathbf{x}_1$. Clearly, no decision vector Pareto dominates $\mathbf{x}_3$, and $\mathbf{x}_3$ can be defined as a Pareto optimal decision vector. Hence, the set of Pareto optimal decision vectors should be determined to obtain solutions of a MOP.

Furthermore, a Pareto optimal decision vector $\mathbf{x}^*$ exists only if no other decision vector $\mathbf{x}$ can dominate it to obtain $y(\mathbf{x}) \preceq y(\mathbf{x}^*)$. And a Pareto optimal set X^* is consisted of all Pareto optimal decision vectors $\mathbf{x}^*$, which can be defined by:

$$X^* = \{\mathbf{x}^* \in \mathcal{X} | \nexists \mathbf{x} \in \mathcal{X} : y(\mathbf{x}) \preceq y(\mathbf{x}^*)\}, \tag{2.7}$$

where $\mathcal{X}$ is a decision space containing a set of feasible decision vectors. Moreover, the Pareto front $\mathcal{F}^*$ regarding the Pareto optimal set X^* is expressed as:

$$\mathcal{F}^* = \{\mathbf{y} = (y_1(\mathbf{x}^*), ..., y_k(\mathbf{x}^*)) | \mathbf{x}^* \in X^*\}. \tag{2.8}$$

Figs. 2.10 and 2.11 show the illustrations of the Pareto optimal set and Pareto front of two exampled functions: $y_1 = \sqrt{1+x^2}$ and $y_2 = 4 + 2\sqrt{1+(x-1)^2}$.

To be specific, the mathematical programming technologies of the MOPs can be classified based on the different ways of searching for solutions. In the following, two classic MCDM approaches are introduced, in which decision makers are needed to deduce the preference choice after a set of search processes.

(1) Simple Additive Weighting (SAW) Method

The basic idea of the SAW method is to allocate a weighting coefficient to each objective function, aiming to minimize or maximize the weighted sum of all objectives. Accordingly, a multi-objective optimization problem can be transformed into a new single objective optimization problem.

Generally, the new problem can be formulated as:

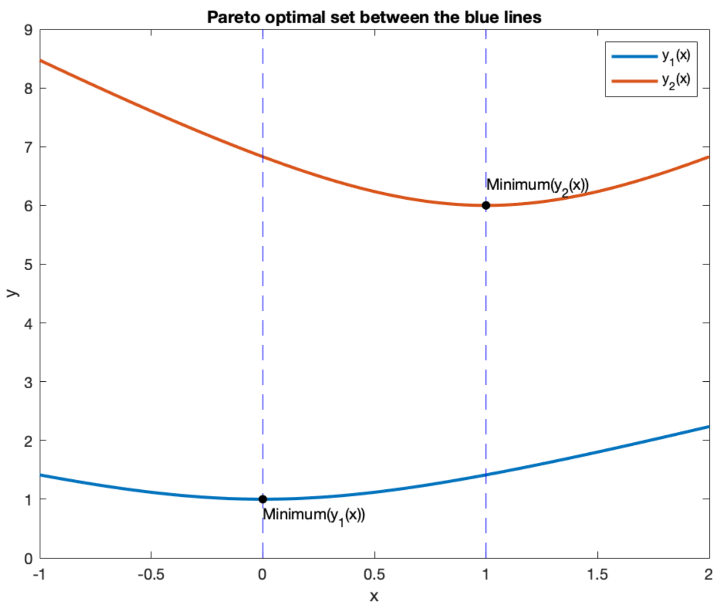


Fig. 2.10 Illustration of the Pareto optimal set.

$$\text{Optimize} \quad \sum_{i=1}^{n} w_i y_i(\mathbf{x}), \tag{2.9}$$

subject to

$$\begin{aligned} f(\mathbf{x}) &= 0, \\ g(\mathbf{x}) &\leq 0, \end{aligned} \tag{2.10}$$

where $w_i \geq 0$ $(i = 1, ..., n)$ and $\sum_{i=1}^{n} w_i = 1$. At least one weighting coefficient must be positive for its corresponding objective function. The set of non-dominated solutions can be obtained by adjusting the weighting coefficient of each objective function. Particularly, if all of weighting coefficients are positive, i.e, $w_i > 0$ for $\forall i$, then the obtained solutions of a SAW problem are Pareto optimal [52].

(2) ε-Constraint Method

In this method, one of the objective functions should be minimized or maximized while the other objectives are considered as constraints limited by the parameters ε. A MOP problem can be transformed into a ε-constraint problem, which is expressed as:

$$\text{Optimize} \quad y_j(\mathbf{x}), \tag{2.11}$$

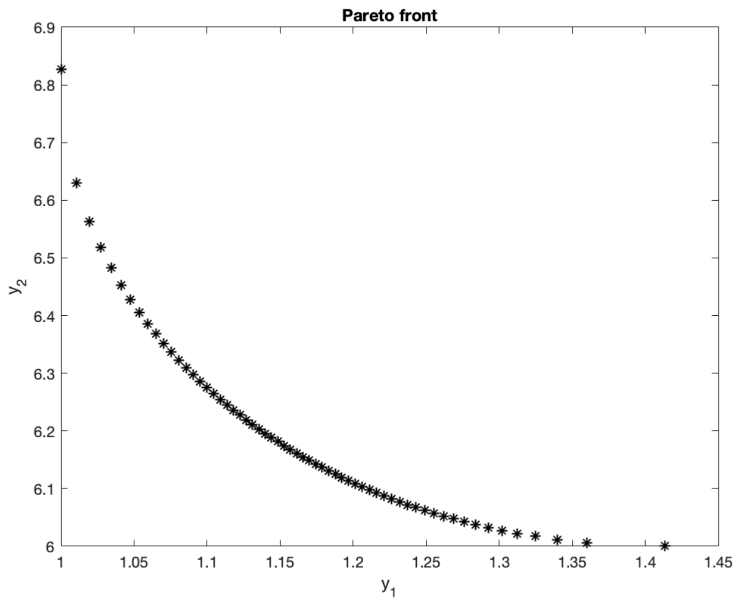


Fig. 2.11 Illustration of the Pareto front.

subject to

$$\begin{aligned} & y_i(\mathbf{x}) \leq \varepsilon_i \quad \forall i = 1, ..., n, \quad i \neq j, \\ & f(\mathbf{x}) = 0, \\ & g(\mathbf{x}) \leq 0. \end{aligned} \tag{2.12}$$

Specifically, the method can generate several Pareto optimal solutions by setting different values of ε_i. The solution $\mathbf{x}^*$ is Pareto optimal if and only if $\varepsilon_i = y_i(\mathbf{x}^*)$ for $\forall i = 1, ..., n, i \neq j$, and $\mathbf{x}^*$ should be an optimal solution for $\forall j = 1, ..., n$ [52]. As a consequence, n or less than n single optimization problems should be solved to generate the Pareto solutions using the ε-constraint approach.

Although the above two mathematical programming methods have been applied to solve the MOPs [53], challenges still exist. For instance, some MOPs belonging to the complex COPs are known as NP-hard, which will take unreasonable computational cost to solve. Besides, their programming models have to be run many times to determine a Pareto optimal set, and extra domain information is usually required.

To solve the challenges, the EMO algorithms have been applied to solve the MOPs widely, which are suitable to solve the complex COPs [54]. The evolutionary algorithms always start with a set of solutions called an initial population, which are generated randomly. The descendant populations are then produced by a series of operators such as mutation, crossover and selection based on the initial population.

In the thesis, the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm [55] is adopted to solve the NSDP with multiple design objectives. The MOPSO is an extended version of the Particles Swarm Optimization (PSO) algorithm [56], which is a well-known meta-heuristic optimization method for the MOPs. The PSO algorithm is an emerging approach based on the evolutionary algorithms, which was first proposed by Kennedy and Eberhart in 1995. The PSO algorithm is inspired by the bird herd behaviour and it leverages the swarm intelligence to obtain the optimal solutions. To be specific, a swarm of particles represented as potential solutions fly through the search space following the current optimum particles to search for the optimal solution with a certain velocity.

Each particle i is associated with two vectors: the position vector $X_i = (x_i^1, x_i^2, ..., x_i^D)$ and the velocity vector $V_i = (V_i^1, V_i^2, ..., V_i^D)$, where D denotes the dimensions of the solution space. The quality of positions of particles can be determined by a fitness function that is adjusted according to different application scenarios. The velocity vectors of particles can be determined by three factors: the position with the best fitness found so far for the particle i $(pBest_i)$, the best position in the swarm $(gBest_i)$ and the current position of the particle i.

During the evolutionary process, the position and velocity of the particle i on the d^{th} dimension can be updated as follows:

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{2.13}$$

$$v_i^d(t+1) = w \cdot v_i^d(t) + c_1 \cdot r_1^d(pBest_i^d) + c_2 \cdot r_2^d(gBest_i^d), \tag{2.14}$$

where w denotes the inertia weight, c_1 is the cognition weight and c_2 is the social weight. r_1^d and r_2^d are two random variables uniformly distributed in the range of $[0,1]$.

The flowchart of the MOPSO algorithm is shown as Fig. 2.12. Firstly, initialize the population POP for each particle $POP[i]$ $(i = 0, ..., Max)$, where Max is the maximum number of the particles. Then, initialize the velocity of each particle and evaluate them.

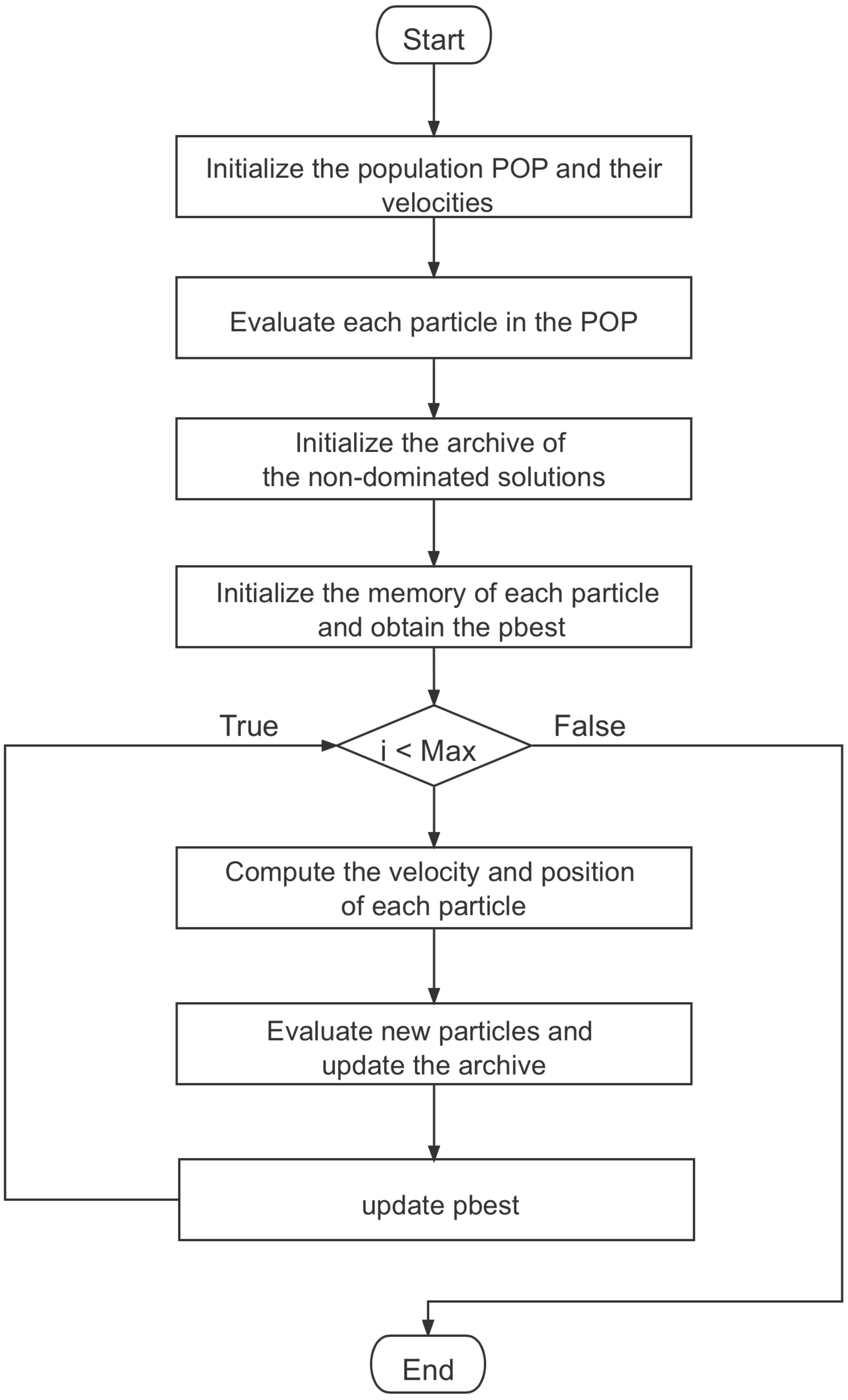

Start
Initialize the population POP and their velocities
Evaluate each particle in the POP
Initialize the archive of the non-dominated solutions
Initialize the memory of each particle and obtain the pbest
True
i < Max
False
Compute the velocity and position of each particle
Evaluate new particles and update the archive
update pbest
End

Fig. 2.12 Flowchart of the MOPSO algorithm.

And store the particles positions of non-dominated solutions into the archive. Locate each particle in the search space by using hyper-cubes and initialize the memory of each particle to store the $pBest_i$. Next, compute the positions and velocities of new particles. And evaluate each particle in POP and update the elements of the archive of the non-dominated vectors. If the particle's current position has higher quality than the position stored in its memory, then update the particle's new position. Repeat the above procedures until all of iteration processes end. An example of the iteration processes of the MOPSO algorithm is illustrated in Fig. 2.13.

Robust Optimization

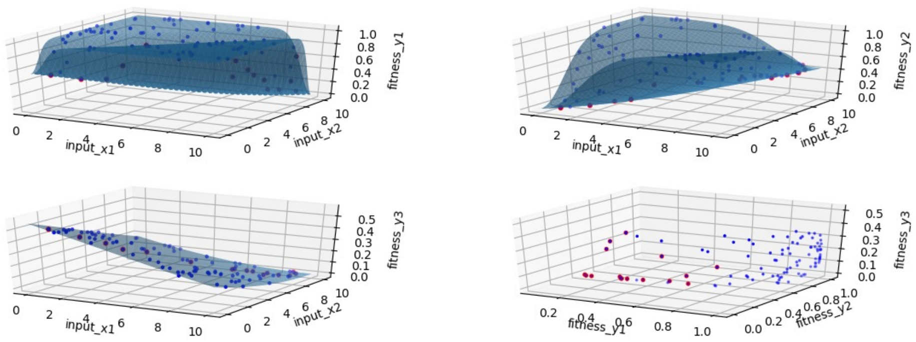
In practice, uncertain data is usually considered in optimization problems. Recently, two main methodologies have been proposed to address data uncertainty, namely Stochastic Optimization (SO) and Robust Optimization (RO) [57]. In stochastic optimization, it is assumed to know the true probability distribution of the uncertain data, however, it is difficult to capture the distribution in practical scenarios. Thus, the robust optimization has been investigated as another important approach for addressing the optimization problems under uncertainty, which can be computationally tractable for many kinds of problems. Robust optimization does not require the probability distributions, but instead, it aims to obtain candidate solutions that are feasible for any realization of the data from an uncertainty set.

In general, for an uncertain linear optimization problem, the formulation can be expressed as:

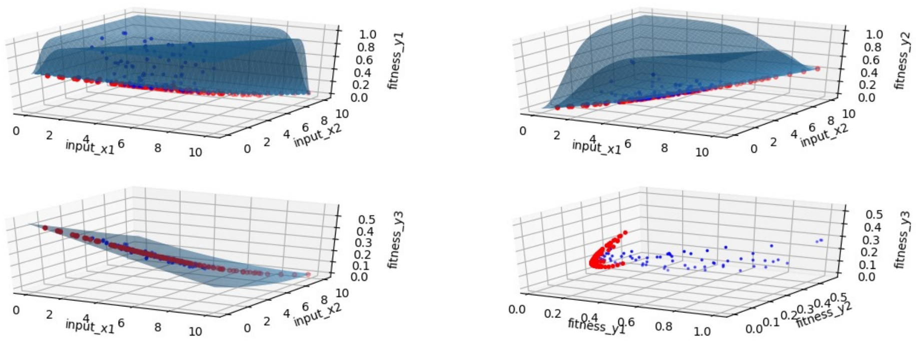
$$\text{Optimize} \quad \sum_j c_j x_j \tag{2.15}$$

subject to

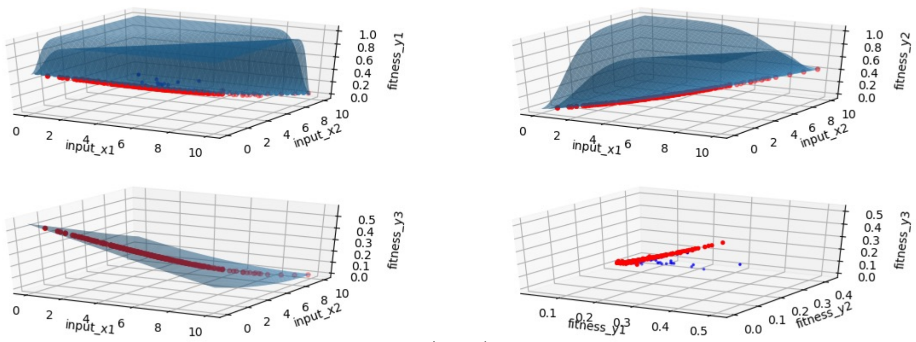
$$\begin{aligned} &\sum_j a_{ij} x_j \leq b_i, \quad \forall i, \\ &\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \end{aligned} \tag{2.16}$$



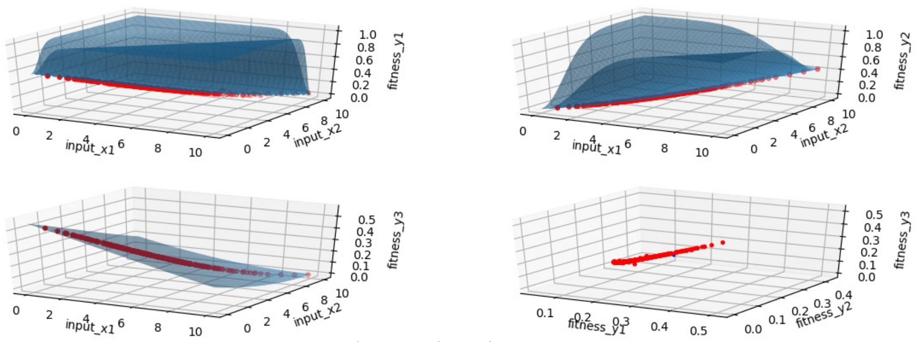
a. Iteration time=1



b. Iteration time=10



c. Iteration time=20



d. Iteration time=30

Fig. 2.13 Illustration of the iteration processes of the MOPSO algorithm.

where $x_j \in \mathbf{x}$ indicates a decision variable, and $c_j \in \mathbf{c}$, $a_{ij} \in \mathbf{A}$ and $b_i \in \mathbf{b}$ denote the uncertain coefficients respectively.

Specifically, the structure of the uncertain set U has a strong effect on the robustness of solutions. Define an uncertain element a_{ij} in U as $a_{ij} = \bar{a}_{ij} + \xi_{ij}\hat{a}_{ij}$, where $\bar{a}_{ij}$ is the nominal value of an uncertain data and $\hat{a}_{ij}$ represents a constant perturbation. ξ_{ij} indicates an independent random variable that is related to the uncertain data [58].

The constraint of (2.16) can be reformulated as:

$$\sum_j a_{ij}x_j + \sum_{j \in J_i} \xi_{ij}\hat{a}_{ij}x_j \le b_i. \tag{2.17}$$

To immunize against the infeasibility of solutions for any ξ in U, the above formula can be rewritten as:

$$\sum_j a_{ij}x_j + max_{\xi \in U}\{\sum_{j \in J_i} \xi_{ij}\hat{a}_{ij}x_j\} \le b_i. \tag{2.18}$$

In the following, three types of uncertain sets are introduced, where Ψ and Γ are two adjustable parameters that can control the size and structure of the uncertainty sets.

- Box Uncertainty Set:

  $U_\infty = \{\xi | \|\xi\|_\infty \le \Psi\} = \{\xi || \xi_j| \le \Psi, \forall j \in J_i\}$.

  And its corresponding robust counterpart constraint is shown as:

$$\begin{cases} \sum_j a_{ij}x_j + \Psi \sum_{j \in J_i} \hat{a}_{ij}u_j \le b_i \\ -u_j \le x_j \le u_j. \end{cases} \tag{2.19}$$

- Polyhedral Uncertainty Set

  $U_1 = \{\xi | \|\xi\|_1 \le \Gamma\} = \{\xi | \sum_{j \in J_i} |\xi_j| \le \Gamma\}$.

  The corresponding robust counterpart constraint is expressed as:

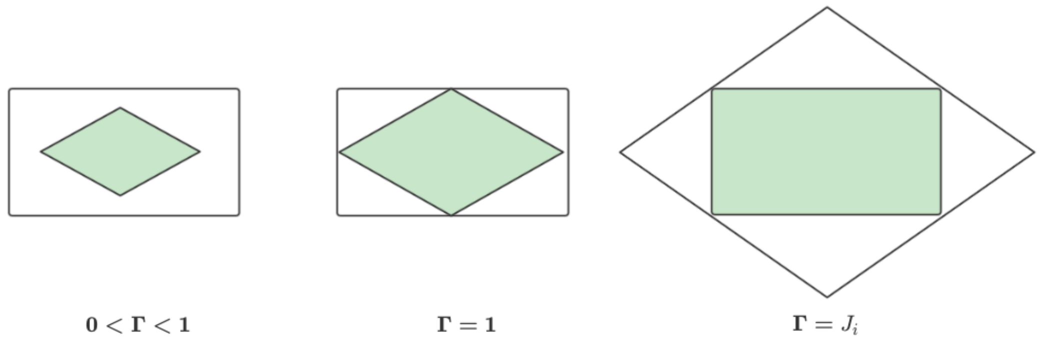


Fig. 2.14 Illustration of the structures of the uncertain set with different values of Γ.

$$\begin{cases} \sum_j a_{ij}x_j + \Gamma s_i \leq b_i \\ s_i \geq \hat{a}_i u_j, \forall j \in J_i \\ -u_j \leq x_j \leq u_j. \end{cases} \tag{2.20}$$

- Box & Polyhedral Uncertainty Set:

  $U_{1\infty} = \{\xi | \sum_{j \in J_i} |\xi_j| \leq \Gamma, |\xi_j| \leq \Psi, \forall j \in J_i\}$.

  Then the corresponding robust counterpart constraint can be formulated as:

$$\begin{cases} \sum_j a_{ij}x_j + \Gamma s_i + \Psi \sum_{j \in J_i} t_{ij} \leq b_i \\ s_i + t_{ij} \geq \hat{a}_i u_j, \forall j \in J_i \\ -u_j \leq x_j \leq u_j, \\ s_i \geq 0, t_{ij} \geq 0. \end{cases} \tag{2.21}$$

  In particular, when $\Psi = 1$, the intersection between the box and polyhedral sets is defined as an overlap set called "box+polyhedral" uncertainty set [58]. The different structures of the overlap set are illustrated in Fig. 2.14 with different values of Γ.

Neural Combinatorial Optimization

Combinatorial optimization (CO) problems aim to identify the optimal solution of an objective function from a finite set of candidate solutions whose domain space is discrete and huge [59]. Usually, the CO problems are relatively easy to solve when their searching space of solutions is small. However, it is difficult to solve the CO problems of large scales because the exact methods cannot be computed with a reasonable computational cost, and meta-heuristics and heuristics algorithms cannot ensure to obtain the optimal solution with a speed convergence.

As a consequence, a novel approach to tackle the Combinatorial Optimization Problems by employing the Reinforcement Learning (RL) and Neural Networks (NNs) has been emerged, named Neural Combinatorial Optimization (NCO) [60]. For instance, Pointer Network is a fundamental NCO approach, which mainly employs the Recurrent Neural Networks (RNNs) architecture in the Deep Learning (DL) methods [61]. The details of RL and DL are introduced in Section 2.4.4.

The NCO method has been proven feasible in finding near-optimal solutions to some classic CO problems, such as TSP and knapsack problems [60]. To be specific, data labels are not necessary during training processes in the NCO since the rewards obtained by learning agents can used to evaluate candidate solutions. And the solution (action) space of CO problems is usually associated with a large-dimensional search space. Thus, the NCO can leverage the policy-based approaches in the RL methods to learn a policy function directly to map an instance of a problem (state) to an action in the environment [37].

2.4.4 Deep Reinforcement Learning

The proposed NSDP is an application of the Combinatorial Optimization problem, which can be solved efficiently using the NCO methodology. Thus, the Deep Reinforcement Learning (Deep RL) technology is regarded as a promising approach for addressing the problem.

For instance, Reference [62] proposed an end-to-end network slicing system, aiming to manage various resources, such as radio spectrum resources, transportation bandwidth and computing resources. Specifically, this work presented a derivative-based optimization framework to achieve the goal by using deep reinforcement learning techniques. Wang *et al.* [63] presented a dynamic resource scheduling framework for network slices. This framework aimed not only to obtain an automatic resource allocation scheme but also to maximize the resource utilization of slices. And a Convolutional Neural Network (CNN) was deployed to model the network environment. State tensors can be transferred into images, which can be fed into the learning agent.

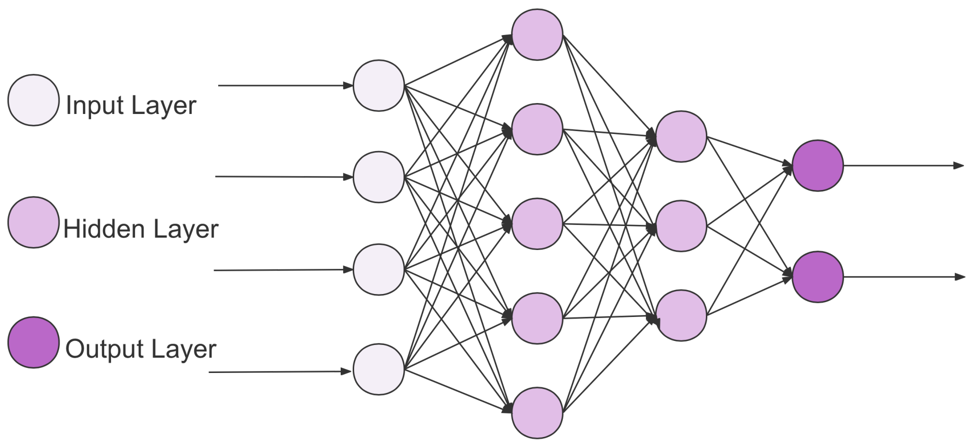
General Idea Behind Deep RL

Reinforcement Learning (RL) [64] is a branch of Machine Learning methods in which a learning agent can learn from interacting with an environment through trials and errors. The RL aims to compute a policy or a control strategy to maximize the expected cumulative reward over time.

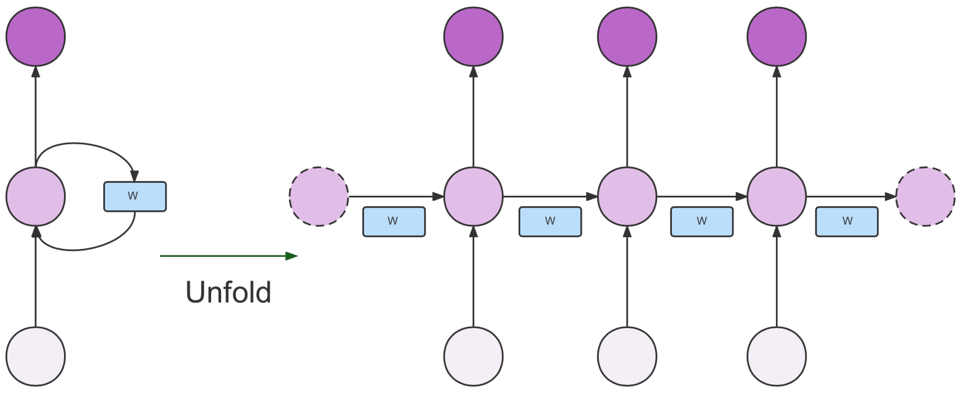
In the RL setting up, a decision-maker, called the learning agent, interacts with the environment built by a set of states $s_t \in \mathcal{S}$ of observing the consequences of actions. The learning agent can take a certain action $a_t \in \mathcal{A}(s_t)$, as a reaction of the current state s_t. After selecting the action a_t at time step t, the agent will receive a scalar reward $r_{t+1} \in \mathcal{R}$ as feedback, and the environment takes a transition to a new state s_{t+1} according to the current state s_t and the chosen action a_t.

The learning agent tries to learn a policy π at each time step, which is a behavior strategy that maps from states to the probability of choosing a possible action. The policy $\pi(s,a) = \mathcal{P}[a_t = a | s_t = s]$ denotes the probability of selecting $a = a_t$ when $s = s_t$. Given a state, the goal of the learning agent is to learn and perform an optimal policy for maximizing the expected reward in the environment.

Furthermore, the Deep Learning (DL) [65] enables the RL to solve the decision-making problems that were previously intractable, such as the problems with high-dimensional states and action spaces. The DL can avoid manual interference of a data structure by automatic learning from the raw data. Besides, the Deep learning approaches with Neural Networks have been applied to dramatically improve the state-of-the-art engineering areas such as the object detection, speech recognition and natural language processing. Any neural network with two or more hidden layers is defined as a Deep Neural Network (DNN). In particular, Feedforward Neural Networks



a. Feedforward Neural Networks (FNNs)



b. Recurrent Neural Networks (RNNs)

Fig. 2.15 Basic architecture of the DNNs: a. FNNs and b. RNNs.

(FNNs) and Recurrent Neural Networks (RNNs) are two typical DNNs models, their basic architectures are illustrated in Fig. 2.15.

To be specific, the Convolutional Neural Networks (CNNs) paradigm is the most well known FNNs model, where the information flows in one direction without cycles or loops, i.e., from the input through hidden layers and to the output [66].

In contrast, the RNN is a kind of recursive Artificial Neural Networks (ANNs), where connections between neurons form directed loops or cycles. The output at each time step depends on the instant inputs and the previous states of neurons. The RNNs compose a set of hidden states and optional outputs, which are appropriate for operating on variable-length sequences. It can determine a probability distribution over a set of sequential data by learning to predict the next symbol in a sequence [67].

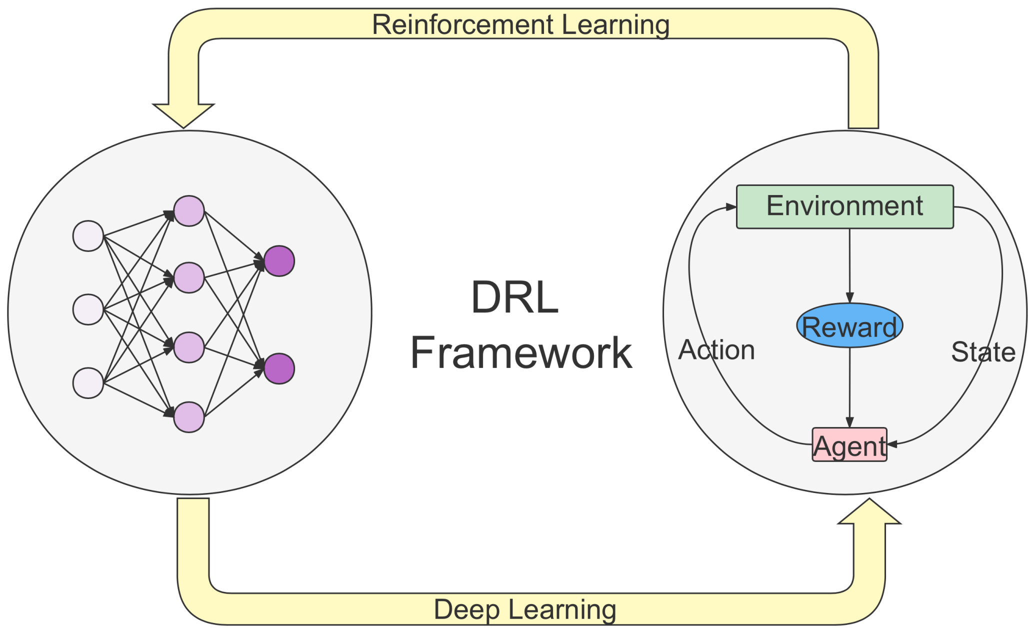


Fig. 2.16 Deep Reinforcement Learning Framework.

Concretely, recurrent neurons are responsible for both performing computation and memorizing information. To solve the long-term sequential problems, the Long Short-Term Memory (LSTM) structure [68] is often applied in the RNNs. The LSTM can solve problems by incorporating memory units to let the network forget the previously vain hidden states.

Consequently, the RL learning framework, in combination with the DNNs model, is proposed as Deep Reinforcement Learning (DRL or Deep RL) [65], which aims to improve the policy performance of the problems with high dimensional raw data input by interacting with the environment.

The basic framework of the Deep Reinforcement Learning is shown in Fig. 2.16. In this figure, the left Deep Learning component receives the target observation information from the environment and gives the state information to the current environment. And the right Reinforcement Learning model produces a corresponding action based on the current state and evaluates the value of its expected reward [69]. Thus, the DRL model enables the learning agent to have a good perception of the environment, which is a very active research area for a wide range of engineering scenarios.

Fundamental Definitions

In this subsection, the fundamental definitions of the DRL are summarized as follows based on References [70], [71], [69].

- Return

  The long-term cumulative return $\mathcal{R}_t$ after time step t with a finite time horizon that terminates at T is defined by:

$$\mathcal{R}_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T = \sum_{i=t+1}^{T} r_i. \tag{2.22}$$

  Moreover, the expected discounted return $\mathcal{R}_t$ is represented as follows:

$$\mathcal{R}_t = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T] = E[\sum_{i=0}^{\infty} \gamma^i r_{t+1+i}], \tag{2.23}$$

  where $E[\cdot]$ indicates the expectation regarding to the return distribution and $0 \le \gamma \le 1$ denotes the discount factor.

- State-Value Function

  Value functions are used to predict the potential future reward and evaluate the benefit that the agent can obtain in a given state.

  The state-value function $V_\pi(s)$ is defined as the expected return when starting in the state s following a policy π, which is shown as:

$$V_\pi(s) = E_\pi[\mathcal{R}_t | s_t = s] = E_\pi[\sum_{i=0}^{\infty} \gamma^i r_{t+1+i} | s_t = s]. \tag{2.24}$$

  The optimal state-value function $V_*(s)$ is associated with the optimal policy π_*. And $V_*(s)$ is defined as follows:

$$V_*(s) = \max_{\pi} V_{\pi_*}(s) \qquad \forall s \in \mathcal{S}. \tag{2.25}$$

If $V_*(s)$ is available, π_* can be obtained by selecting the optimal actions in the action set $\mathcal{A}$ at state s_t, which can maximize $E_\pi[\mathcal{R}_{t+1}|s_{t+1} = s]$.

- Action-Value Function

  The action-value function is the expected return achieved by starting from state s and performing action a and then following a policy π. It is expressed as:

$$Q_\pi(s,a) = E_\pi[\mathcal{R}_t|s_t = s, a_t = a] = E_\pi[\sum_{i=0}^{\infty} \gamma^i r_{t+1+i}|s_t = s, a_t = a]. \tag{2.26}$$

  Similarly to $V_*(s)$, the optimal action-value function $Q_*(s,a)$ is defined by:

$$Q_*(s,a) = \max_\pi Q_{\pi_*}(s,a) \qquad \forall s \in \mathcal{S}, \forall a_t \in A. \tag{2.27}$$

- Optimal Policy

  The optimal policy π_* is the policy that can achieve the best value of the cumulative discounted return. Given the optimal state-value function V_*, the optimal policy π_* can be represented by:

$$\pi_* = \underset{\pi}{argmax} V_*(s) \qquad \forall s \in \mathcal{S}. \tag{2.28}$$

  Besides, similarly, given the optimal action-value function Q_*, the optimal policy π_* can be extracted by:

$$\pi_*(s) = \underset{a \in \mathcal{A}}{argmax} Q_*(s,a) \qquad \forall s \in \mathcal{S}. \tag{2.29}$$

- Markov Decision Process (MDP)

  The MDP [72] can provide a mathematical framework to formalize the sequential decision-making problems in the DRL. A MDP is illustrated in Fig. 2.17.

  In general, a MDP can be represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

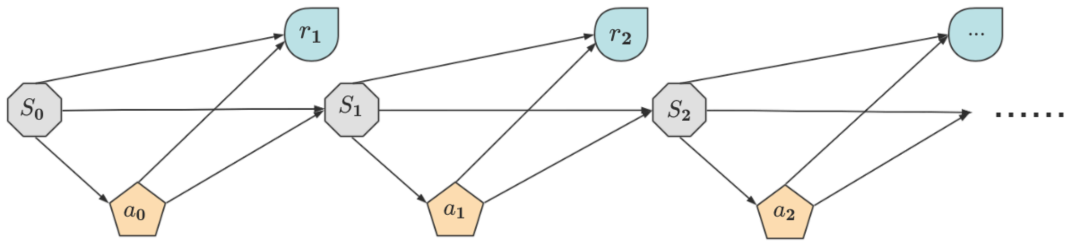


Fig. 2.17 Illustration of Markov Decision Process.

- $\mathcal{S}$ is a finite set of possible states;
- $\mathcal{A}$ is a finite set of available actions;
- $\mathcal{P}$ is a state transition probability function $\mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$. It maps from the state-action pair to the probability distributions of the next state, and follows the following Markov Property:

$$\mathcal{P}(s_{t+1} = s' | s_0, a_0, ..., s_t, a_t) = \mathcal{P}(s_{t+1} = s' | s_t, a_t); \tag{2.30}$$

- $\mathcal{R}$ is a reward function $\mathcal{R} = E[\mathcal{R}_{t+1} | \mathcal{S}_t = s]$;
- γ is a discount factor, $\gamma \in [0, 1]$.

In a MDP, $\mathcal{T} = (s_t, a_t, s_{t+1}, a_{t+1}, \cdots, s_T, a_T)$ is assumed as a trajectory, and (s_t, a_t, r_t, s_{t+1}) is called an experience. The optimal policy π_* can be obtained when $E_{s_{t+1} \sim \mathcal{T}(s_{t+1}|s_t, a_t)}[V_*(s_{t+1})]$ achieves the maximum expected return.

In addition, the action-value function Q_π can be re-expressed as a Bellman equation [73] by exploiting the Markov Property, which is shown as follows:

$$Q_\pi(s_t, a_t) = E_{t+1}[r_{t+1} + \gamma Q_\pi(s_{t+1}, \pi(s_{t+1}))]. \tag{2.31}$$

Value-based Deep RL

Given a Deep RL problem, two main approaches are proposed to compute the optimal policy. One approach is based on searching in the space of value functions, which is called the value-based Deep RL; the other one is based on searching in the space of

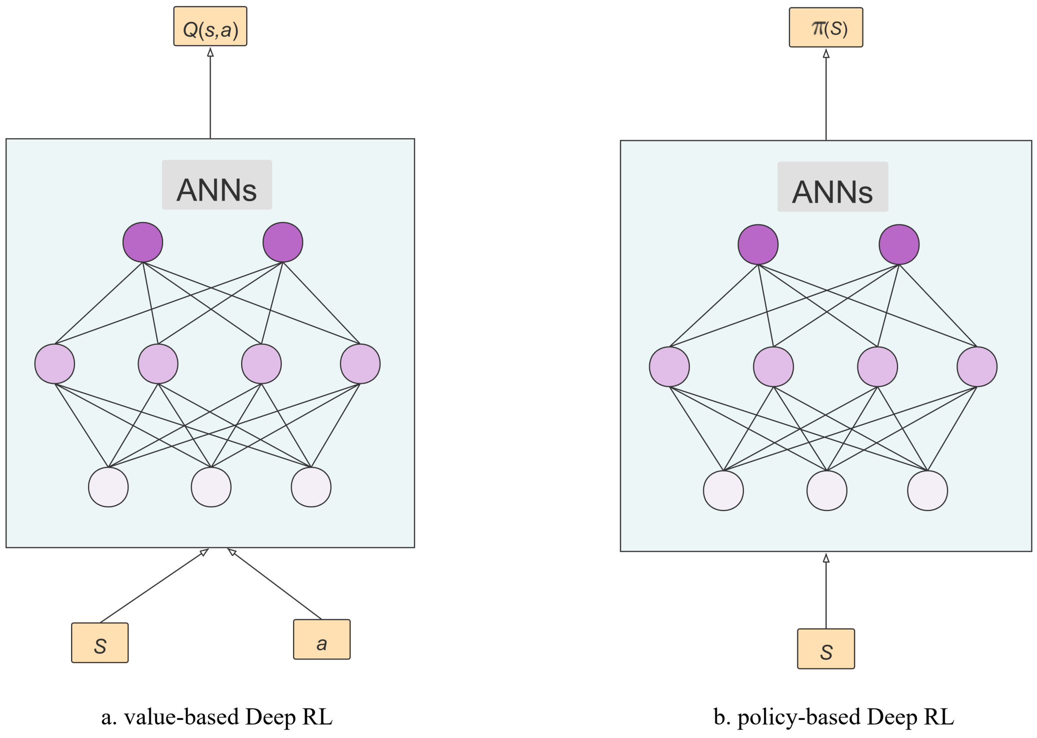


Fig. 2.18 Basic structures of DRL approaches: a. value-based DRL methods, b. policy-based DRL methods.

policies, which is called the policy-based Deep RL [70]. The basic structures of the two types of the DRL are illustrated in Fig. 2.18.

In the value-based DRL algorithms, the learning agents attempt to approximate value functions and deduce the optimal policy π_*, instead of finding an explicit policy. In particular, the problem of searching π_* can be transformed into a problem of approximating the optimal action-value function. The Monte Carlo (MC) estimation methods and Temporal Difference (TD) learning are two typical technologies to achieve the goal of function approximation.

The MC methods learn directly from the return of whole historical experience, which contributes to low bias in estimation. The action-value function is updated periodically to improve the policy based on the principle of Policy Iteration [74]. The process of iteration can be represented as follows:

$$\pi_0 \rightarrow Q_{\pi_0} \rightarrow \pi_1 \rightarrow Q_{\pi_1} \rightarrow ...\pi_k \rightarrow Q_{\pi_k} \rightarrow ...\pi_* \rightarrow Q_{\pi_*}, \tag{2.32}$$

where π_0 represents the initial policy. The current estimation $Q_{\pi_k}(s_t,a_t)$ can be updated based on the cumulative return $\mathcal{R}(\mathcal{T}|s_t,a_t)$, which is shown as:

$$Q_{\pi_k}(s_t,a_t) \leftarrow Q_{\pi_k}(s_t,a_t) + \alpha(\mathcal{R}(\mathcal{T}|s_{t+1},a_{t+1}) - Q_{\pi_k}(s_t,a_t)), \tag{2.33}$$

where α is the learning rate.

It is assumed that the action-value function Q_π can asymptotically approach the real action-value function with respect to the current policy, and the MC methods can converge to the optimal Q_* and π_* eventually. A new greedy policy π_{k+1} can be constructed based on the current Q_{π_k}, where actions are taken greedily to achieve the maximal return, and it can be defined as follows:

$$\pi_{k+1}(s_t) \leftarrow \underset{a\in\mathcal{A}}{argmax}\, Q_{\pi_k}(s_t,a), \quad \forall s_t \in \mathcal{S}. \tag{2.34}$$

In addition, the TD learning method learns from incomplete experience efficiently based on the prediction of the next state, which plays to low variance in estimation. Its iteration process is similar to (2.32). However, differently, the TD learning updates the action-value function by using the Dynamic Programming (DP) method [75] with bootstrapping. The current estimation $Q_{\pi_k}(s_t,a_t)$ is updated based on the immediate return r_{t+1} and the predicted value of the next state, which is expressed as:

$$Q_{\pi_k}(s_t,a_t) \leftarrow Q_{\pi_k}(s_t,a_t) + \alpha(r_t + \gamma \underset{a_{t+1}\in\mathcal{A}}{max} Q_{\pi_k}(s_{t+1},a_{t+1}) - Q_{\pi_k}(s_t,a_t)), \tag{2.35}$$

where $r_t + \gamma \underset{a_{t+1}\in\mathcal{A}}{max} Q_{\pi_k}(s_{t+1},a_{t+1}) - Q_{\pi_k}(s_t,a_t)$ is called the TD error, which has the Markov Property. Thus, the TD learning can learn after each decision epoch [76] rather than finishing the complete episode.

The typical value-based Deep RL algorithms include Deep Q-learning Network (DQN) [77], Double Deep Q-learning Network [78] and Dueling Q-learning Network [79]. For the Deep Q-Learning problems, a DNN function approximator with parameter ω is trained to estimate the Q-values, which is shown as:

$$Q(s,a;\omega) \approx Q_\pi(s,a). \tag{2.36}$$

In general, the approximation process can be achieved by minimizing the loss function of Mean Squared Error (MSE) at each step t:

$$\mathscr{L} = E_{s,a,r,s'\sim\pi}[(y_t - Q(s,a;\omega))^2] \tag{2.37}$$

where $y_t = r + \gamma max_{a'} Q(s',a';\omega)$ is the TD target and $\delta_t = y_t - Q(s,a;\omega)$ is the TD error. To be specific, the Deep Q-learning-based agent learns the optimal policy through a greedy policy and $a = max_a Q(s,a;\omega)$. Usually, the ε-greedy policy is used to ensure a good convergence, which selects a random action with probability ε and the greedy action with probability $1-\varepsilon$.

To overcome the over-estimation problem of the Deep Q-learning algorithms, the Double Deep Q-learning model [78] is introduced to select and evaluate action values simultaneously by using two Q-value networks, such as Q and Q' with parameters ω and ω' respectively. The loss function is defined by:

$$\mathscr{L} = E_{s,a,r,s'\sim\pi}[(r + \gamma Q'(s', \underset{a'}{argmax}\, Q(s',a;\omega),\omega') - Q(s,a;\omega))^2], \tag{2.38}$$

where $\underset{a'}{argmax}\, Q(s',a;\omega) = a_{max}(s',\omega)$.

Further, to solve the MDPs with large action spaces, a dueling architecture is proposed. In the Dueling Deep Q-learning, the value of an action a at state s can be computed by two functions [80]. The first part is the state-value function $\mathscr{V}(s)$, which is used to estimate the benefit of being in a state s. And the second part is the action-value function $\mathscr{A}(s,a) = Q(s,a) - \mathscr{V}(s)$, which can estimate the reward of picking the action a at state s. As a result, the Q-value function can be represented as: $Q(s,a) = \mathscr{V}(s) + \mathscr{A}(s,a)$. Specifically, it can be expressed as follows:

$$Q(s,a;\omega,\sigma,\beta) = \mathscr{V}(s;\omega,\sigma) + (\mathscr{A}(s,a;\omega,\beta) - \frac{1}{|\mathcal{A}|}\sum_{a'\in\mathcal{A}} \mathscr{A}(s,a';\omega,\beta)), \tag{2.39}$$

where σ and β are the parameters of the two functions $\mathscr{V}(s;\omega)$ and $\mathscr{A}(s,a';\omega,\beta)$ respectively, and $|\mathcal{A}|$ is the size of the action space. The loss function of the Dueling Deep Q-learning is formulated in (2.39).

The DQN technologies are mainly applied to deal with the issues that 1) the subsequent states in the Deep RL tasks are correlated, 2) the policy changes frequently due to the slight changes of Q-values [64]. Clearly, the DQN is suitable to be applied in the field of network slicing, because each state in the network slice design environment is related and the slice design plans will be different as the value functions change.

Recently, Qi *et al.* [81] have focused on solving the allocation problem of limited spectrum resources on a finer-grained resolution across network slices based on the Deep Reinforcement Learning techniques. The authors introduced a discrete normalized advantage function into the Deep Q-learning method, trying to separate the Q-value function as a state-value function and an advantage function. And a deterministic policy gradient descent (DPGD) algorithm [82] was adopted to avoid extra calculation of the Q-value for state-action pairs.

Policy-based Deep RL

As shown in (2.32), a policy is inferred from the action-value function in the value-based DRL algorithms. Conversely, the policy-based DRL algorithms aim to model a policy explicitly by employing parameterized approximators, i.e., DNNs. In other words, they do not need to maintain the value functions but directly learn an optimal policy π_*. Thus, for many problems especially with high-dimensional, the policy-based approaches converge and train much more efficiently by avoiding searching the action space.

In the policy-based DRL algorithms, a parameterized policy network π_θ is defined with the parameters θ, which can be updated for maximizing the expected return $E[\mathcal{R}_t|\theta]$. Specifically, a stochastic policy can map from states to the probability distribution over the action space, which is represented as follows:

$$\pi_\theta(s,a) = \mathcal{P}(a|s,\theta). \tag{2.40}$$

The objective function of the policy-based DRL approaches is defined by:

$$\begin{aligned} J(\theta) &= E_{\pi_\theta}[\mathcal{R}(\mathcal{T})] \\ &= \sum_{\mathcal{T}} \mathcal{P}(\mathcal{T}|\theta)\mathcal{R}(\mathcal{T}), \end{aligned} \tag{2.41}$$

where $\mathcal{P}(\mathcal{T}|\theta)$ is the probability of $\mathcal{T}$ with respect to the policy π_θ.

The fundamental idea of the policy-based DRL algorithms is to find the optimal θ_* that can maximize $J(\theta)$, which is shown as:

$$\theta_* = argmax_\theta J(\theta). \tag{2.42}$$

Based on the policy gradient theorem [83], a general form of the policy gradient for π_θ is shown as:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{\mathcal{T}} \nabla_\theta \mathcal{P}(\mathcal{T}|\theta)\mathcal{R}(\mathcal{T}) \\ &= \sum_{\mathcal{T}} \mathcal{P}(\mathcal{T}|\theta)\mathcal{R}(\mathcal{T})\nabla_\theta log\mathcal{P}(\mathcal{T}|\theta) \\ &= E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)Q_{\pi_\theta}(s,a)]. \end{aligned} \tag{2.43}$$

In particular, a classic policy-based DRL algorithm based on the MC policy gradient is proposed called the REINFORCE [83]. The algorithm updates the parameters θ based on the stochastic gradient ascent mechanism and approximates the optimal policy using MC sampling. And it uses the return r_t as an unbiased sample of the action-value function $Q_{\pi_\theta}(s_t, a_t)$. The gradient policy of the REINFORCE is expressed as follows. And its pseudocode is shown as Algorithm 1.

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)r_t]. \tag{2.44}$$

In addition, another famous Deep RL architecture, called the Actor-Critic (AC) algorithm [84], is proposed to solve problems by combining the value-based DRL and policy-based DRL. Because the two methods both have their drawbacks and flaws. Specifically, the policy-based DRL approaches tend to fall in local optima and may

Algorithm 1 REINFORCE Algorithm

```
1: Initialize $\theta$ arbitrarily;
2: for each $\mathcal{T} \sim \pi_\theta$ do
3:     for t=1 to T-1 do
4:         $\theta \rightarrow \theta + \nabla_\theta log\pi_\theta(a|s)v_t$;
5:     end for
6: end for
7: return $\theta$
```

result in a high variability gradient. Besides, the value-based DRL approaches are intractable for the problems with a high dimensional action space.

Consequently, the Actor-Critic architecture combines the strengths of these two approaches. First, the value-function approximator with the parameter ω is used as a critic to estimate the action-value function, which is shown as:

$$Q_\omega \approx Q_{\pi_\theta}(s,a). \tag{2.45}$$

Moreover, the parameterized policy network is used as an actor to update the parameters θ under the guidance of the critic. The approximated policy gradient of the Actor-Critic algorithm is defined by:

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)Q_\omega(s,a)]. \tag{2.46}$$

The Actor-Critic architecture can reduce the variance of gradient, and it is sample-efficient. This architecture has been widely developed into many algorithms, such as Actor-Critic [84], Advantaged Actor-Critic (A2C) [85], Asynchronous Advantaged Actor-Critic (A3C) [86] and Deterministic Policy Gradient (DPG) [87]. The algorithm of the basic Actor-Critic is shown in Algorithm 2.

So far, the policy-based DRL algorithms have gradually been applied in the network slicing research area. For example, in Reference [88], a resource allocation problem in network slicing was proposed, which aimed to maximize the total throughput by using the Soft Actor-Critic algorithm [89]. The work built a mixed action space including discrete and continuous actions, and it satisfied both the energy and queue length constraints of each network slice. Besides, a Lagrangian multiplier was introduced to

Algorithm 2 Actor-Critic Algorithm

```
1:  Initialize s, $\theta$;
2:  sample an action $a \sim \pi_\theta$;
3:  for each step do
4:      sample reward $r = \mathcal{R}$;
5:      sample transition $s' \sim \mathcal{P}$;
6:      sample action $a' \sim \pi_\theta(s', a')$;
7:      $\delta = r + \gamma Q_\omega(s', a') - Q_\omega(s, a)$;
8:      $\theta = \theta + \nabla_\theta log\pi_\theta(a|s) Q_\omega(s, a)$;
9:      $\omega \leftarrow \omega + \delta\phi(s, a)$;
10:     $a \leftarrow a', s \leftarrow s'$.
11: end for
```

deal with the energy and queue constraints during the policy learning processes. Li *et al.* [90] provided an intelligent resource management mechanism in the RAN network slicing scenario based on the A2C algorithm [86]. The mechanism aimed to solve the problem that considered varying service demands regarding to user mobility. And it also tried to make appropriate resource allocation decisions in a dynamic environment for network slicing.

2.5 Conclusions

In this chapter, the technical background of the network slicing design problem is summarized in detail. Specifically, the fundamental concepts of End-to-End network slicing technology are introduced, including the basic knowledge of Virtualization techniques, the differences between virtual machines and containers, the importance of NFV, SDN, cloud and edge computing and the isolation issues in network slicing. Then the principles of deploying and implementing network slices are listed, and the overall framework of network slicing is illustrated. The detailed characteristics of different 5G use cases are also stated, such as eMBB, mMTC and uRLLC. Besides, the research progresses of two prerequisite technologies for achieving the network slicing design policy are introduced, including the VNE and SFCP. And two critical enabling technologies, optimization algorithms and deep reinforcement learning methods, for solving the network slicing design problem are explained.

Technically, the network slicing design problem is derived and developed from the two prerequisite technologies VNE and SFCP. However, the problem can not be solved thoroughly by only using these two techniques. Because there are still several research gaps. For instance, the existing methods in these two technologies mainly focused on the reduction of capital and operational expenditures for deploying one type of virtual networks. These methods ignore the inherent and crucial characteristics of the deployment of 5G network slices, such as the resource utilization efficiency for deploying various network slices, the energy consumption of the network and the different service requirements of diverse 5G use cases and application scenarios. Thus, in the following chapters, the contributions of the thesis are illustrated based on the above-introduced background knowledge, which aims to bridge the research gaps.

Chapter 3

A Basic Design Policy of End-to-End Network Slicing

3.1 Introduction

In this chapter, a basic design policy for network slicing is proposed to address the NSDP problem, which aims to make efficient utilization of network resources in different 5G use cases. An ILP problem is formulated with multiple design objectives for eMBB, mMTC and uRLLC use cases. However, the proposed formulation of the slicing process is mathematically known as a VNE problem, which is proven to be NP-Hard [91]. The characteristic of NP-hard of a VNE problem can be proven by reducing it to the Multiway Separator Problem [92], which means that the problem is hard to be solved in polynomial time. Thus, a heuristic algorithm of the basic design policy is presented as a trade-off solution. The algorithm can implement network slices through a set of efficient iterations using the PSO algorithm [56]. The main contributions of this chapter can be summarized as follows:

- The network model of the basic NSDP is built, including the mathematical models of the physical network and different network slice requests, by adopting the Formal Concept Analysis (FCA) methodology.

- Multiple design objectives of the basic NSDP are presented for the eMBB, mMTC and uRLLC use cases. An ILP formulation of the problem is presented, which aims to utilize the system resources efficiently.
- A heuristic algorithm is proposed inspiring from the PSO algorithm, which can solve the basic NSDP by obtaining sub-optimal solutions. The present algorithm is verified by a set of simulations, which illustrates that its resource efficiency achieves better performance than the existing algorithms.

The rest of this chapter is organized as follows. Section 3.2 gives a brief summary of the related works of network slicing. In Section 3.3, the network model is presented, and the problem description is stated. The ILP formulation of the basic NSDP and the details of the proposed heuristic algorithm are respectively illustrated in Sections 3.4 and 3.5. In Section 3.6, numerical results of the simulation are discussed. Finally, section 3.7 concludes this chapter.

3.2 Related Works

Recently, numerous researches have been taken by industries and academia to explore how to implement network slices in 5G networks. Extensive efforts have been carried to investigate the network slicing technology, which are involved in several research orientations, such as the framework of network slicing, deployment schemes and reconfiguration strategies.

For instance, Taleb *et al.* are first to consider End-to-End network slicing and introduce the PERMIT slice framework [93]. Zhou *et al.* illustrated a Network Slicing as a service (NSaaS) and give an introduction about the business model of NSaaS [94]. And a heuristic algorithm was introduced by Reference [95] for the network slicing deployment considering the complex network theory. A hybrid slice reconfiguration framework and a dimension slice reconfiguration scheme were illustrated to reduce the overhead of network slice reconfiguration in Reference [96]. Reference [97] presented a network slicing problem to minimize the network latency, and a SDN based architecture was proposed to enable the creation of radio and transport slices by predicting slice capacities and congestion with Machine Learning technology. An optimization problem of the network slice provisioning was formulated in Reference [98], which joined the flexible radio access functional splitting methods with the data-plane and control-plane network function sharing polices. Yin *et al.* [99] proposed a prediction-based dynamic network slicing algorithm, which could reallocate the isolated resources of existing network slices by leveraging traffic predictions. The authors attempted to solve the problem of deploying isolated network slices under a dynamic service provision with time-changing network status in Fi-Wi networks.

3.3 Network Model of Basic Network Slicing Design Problem

In this section, the network models of the physical network and network slicing requests are presented by using the FCA methodology, which aims to provide accurate descriptions about the binary relations between network nodes and their heterogeneous attributes. The FCA methodology has been applied widely to analyze binary relations in various domains, which can model concepts and extract rules [100]. Specifically, a formal context can be represented as a triple $(\mathcal{O},\mathcal{A},\mathcal{R})$ for describing the binary relations $\mathcal{R}$ ($\mathcal{R} \subseteq \mathcal{O} \otimes \mathcal{A}$) between an object $\mathcal{O}$ and its attributes $\mathcal{A}$.

3.3.1 Physical Network Model

The physical network is represented as an undirected graph $G = (P, L, A_P, A_L)$, where P represents the set of physical nodes, L represents the set of physical links. A_P and A_L denote the sets of attributes of physical nodes and links, respectively. Let the triple $\mathbb{T} = (P, A_P, S)$ describe the binary relations between P and the set of heterogeneous attributes A_P. Each physical node $p_i \in P$ is associated with its attribute set A_{p_i}. The relation is represented by $S_i \subseteq p_i \otimes A_{p_i}$ $(S_i \in S)$. A_P can be represented as $A_P = (\{c_i, m_i\} | p_i \in P)$, where c_i and m_i denote the levels of CPU and memory of p_i, respectively. The binary relations between p_i and its attributes can be denoted by $(p_i, A_{p_i}) \in S_{p_i}$.

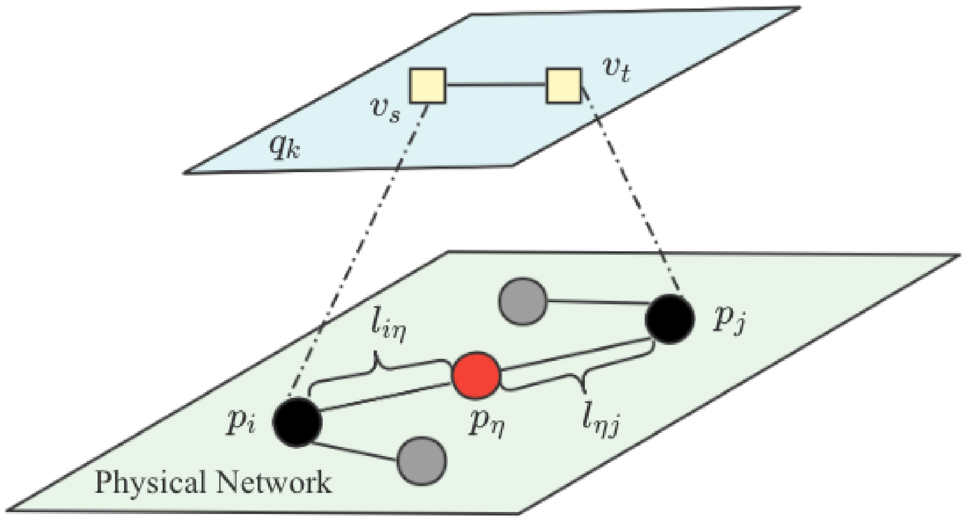


Fig. 3.1 Illustration of a physical path $\mathcal{L}_{ij}$ including a set of physical links.

Moreover, a physical link $l_{i'j'}$ directly connecting two physical nodes $p_{i'}$ and $p_{j'}$ is associated with the bandwidth capacity $b(l_{i'j'})$ and base delay attribute $\delta(l_{i'j'})$. A physical path $\mathcal{L}_{ij}$ is composed of a set of physical links, which can be defined by $\mathcal{L}_{ij} = \{l_{i\eta}, p_\eta, l_{\eta j}\}$. To be specific, p_η is a forwarding node, and $\mathcal{L}_{ij}$ is illustrated in Fig. 3.1. The bandwidth capacity of $\mathcal{L}_{ij}$ is represented by $b(\mathcal{L}_{ij}) = \min(b(l_{i\eta}), b(l_{\eta j}))$. The base delay attribute of the physical path $\mathcal{L}_{ij}$ is defined by $\delta(\mathcal{L}_{ij}) = \sum \delta(l_{i'j'}), l_{i'j'} \in \mathcal{L}_{ij}$. And $h(\mathcal{L}_{ij})$ is the minimum hops among the set of physical links. Particularly, $A_L = (b(\mathcal{L}_{ij}), \delta(\mathcal{L}_{ij}), h(\mathcal{L}_{ij}))$.

The binary relation between each physical node and its attributes can be expressed as a formal context, which is illustrated in Table 3.1. For this given example, the physical network contains 6 physical nodes, and they have three different levels of CPU and memory capacities. In addition, the bandwidth capacity of physical nodes refers to the total bandwidth resources of their adjacent physical paths.

Table 3.1 Formal context of an example physical network

Node	CPU Level			Memory Level			Bandwidth
	lev_1	lev_2	lev_3	lev_1	lev_2	lev_3	
p_1	×			×			×
p_2		×				×	×
p_3		×				×	×
p_4			×		×		×
p_5	×			×			×
p_6			×	×			×

Table 3.2 Formal context of a given network slice request

Node	CPU Level			Memory Level			Bandwidth
	lev_1	lev_2	lev_3	lev_1	lev_2	lev_3	
v_1^k		×		×			×
v_2^k	×					×	×
v_3^k		×			×		×

3.3.2 Network Slicing Request Model

Let $\mathbb{Q}$ represent the set of network slice requests, which consists of three types of use cases. $\mathbb{Q}$ is defined by $\mathbb{Q} = (\mathbb{Q}_e \cup \mathbb{Q}_m \cup \mathbb{Q}_u)$, where $\mathbb{Q}_e$, $\mathbb{Q}_m$ and $\mathbb{Q}_u$ denote the sets of eMBB, mMTC and uRLLC use cases, respectively.

Similar to the physical network, each network slice request is defined by an undirected graph $Q = (V, E, I_V, I_E, \mathcal{T})$, where V denotes the set of requested Virtual Network Functions (VNFs), and E indicates the requested virtual links of network slice requests. I_V and I_E respectively represent the resource requirements of VNFs and virtual links regarding to the resource capacities of physical nodes and links. In addition, $\mathcal{T}$ denotes the types of network slice requests.

Let a triple $\mathbb{R}_k = (O_k, I_k, F_k)$ represent the binary relationships of a network slice request q_k, in which O_k denotes the set of requested objects, including the set of VNFs v and the set of virtual links e. The requirements of q_k are represented as $I_k = (I_v, I_e)$, where $I_v = (\{c_{v_s}, m_{v_s}\} | v_s \in v)$ and $I_e = (\{b(e_{st})\} | e_{st} \in e)$. Specifically, c_{v_s} and m_{v_s} represent the levels of requested CPU and memory of the VNF v_s respectively, and $b(e_{st})$ denotes the requested bandwidth of the virtual link between the VNFs v_s and v_t. The binary relationship between v and I_v are denoted by $F_k \subseteq v \otimes I_v$ $(v \in V)$. A given example of a network slice request is illustrated in Table 3.2. As can be seen from this table, different VNFs may require various levels of CPU and memory resources.

3.3.3 Problem Description

In this chapter, the primary goal of the basic NSDP is to achieve the utilization efficiency of physical resources while satisfying the diverse service requirements of network slice requests by deploying them onto a shared physical infrastructure. Solving the NSDP problem is mainly to determine a set of design operations under the premise of meeting the different service requirements of each network slice request. The design operation of each network slice is composed by two phases: 1) VNFs are mapped onto physical nodes, 2) virtual links are mapped onto physical paths.

The proposed network slicing design policy consists of a set of design operations, which is defined by $\mathcal{D} = (\mathcal{D}^1, \mathcal{D}^2, \cdots, \mathcal{D}^{\mathcal{K}})$, where $\mathcal{D}^k (\forall k \in \mathcal{K})$ is the set of design operations for a network slice request q_k. Resource requirements of different network slices must be satisfied to ensure the Quality of Services for various use cases. To be specific, $\mathcal{D}^k$ is represented by $\mathcal{D}^k = (\mathcal{D}^k_{v_s}, \mathcal{D}^k_{e_{st}})$:

$$\mathcal{D}^k = \begin{cases} \mathcal{D}^k_{v_s} : [v_s, I_{v_s}] \to [p_i, A_{p_i}] & v_s \in v, p_i \in P \\ \mathcal{D}^k_{e_{st}} : \ [e_{st}, d_k] \to [\mathcal{L}_{ij}, A_{\mathcal{L}_{ij}}] & e_{st} \in e, \mathcal{L}_{ij} \in L \end{cases}, \tag{3.1}$$

where $\mathcal{D}^k_{v_s}$ indicates that v_s is placed on p_i. And its resource requirements can be satisfied by the resource capacities of p_i. Similarly to $\mathcal{D}^k_{e_{st}}$, the bandwidth resources of $\mathcal{L}_{ij}$ are consumed to process the traffic demands of e_{st}. Overall, the task of solving the present NSDP can be transformed into obtaining a set of $\mathcal{D}^k$, which simplifies the difficulty of solving the proposed design problem.

We assume that one VNF of each network slice request can only be mapped onto one physical node in the physical network, and a physical node can only host at most one VNF from a same request. Besides, an illustration of the basic NSDP is shown in Fig. 3.2, in which three different types of network slice requests can be deployed onto a shared physical network. For instance, the VNFs a, b and c of an eMBB slice are deployed onto the physical nodes A, B and C respectively, and the virtual links between them are placed onto the physical path $A \to B \to C$. Similarly, mMTC slices and uRLLC slices are also deployed onto the same underlying physical network.

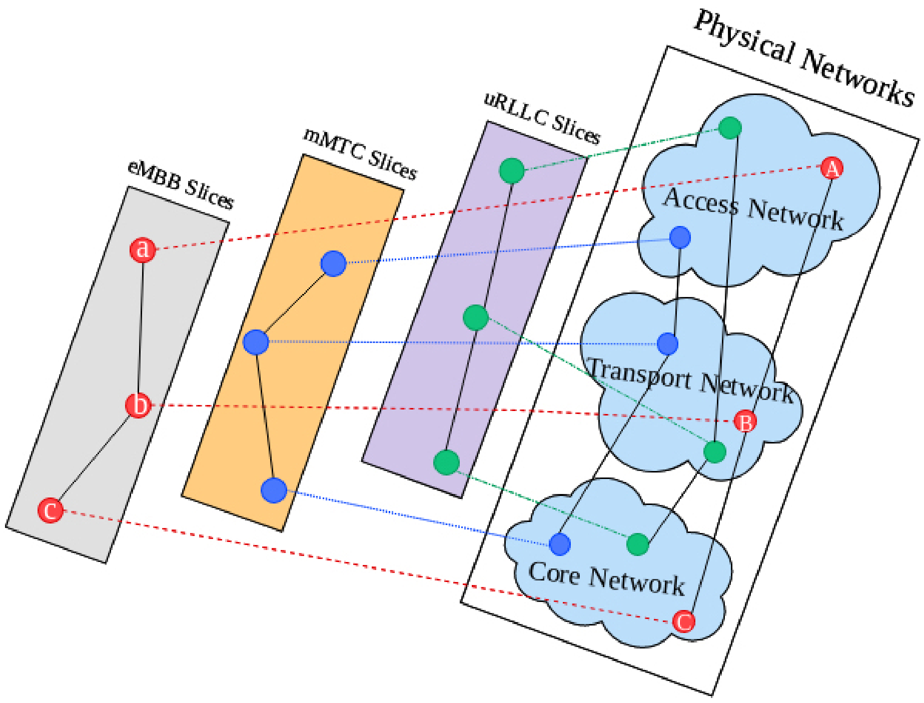


Fig. 3.2 Illustration of the basic network slicing design problem

3.4 Problem Formulation

In this section, an ILP model of the basic NSDP is formulated with multiple design objectives, which aims to provide network slicing services for different types of application scenarios.

The details of the proposed ILP formulation for the basic NSDP are shown as follows.

Variables:

- Let decision variable $\rho_i^{s,k} \in \{0,1\}$ take 1 if the VNF v_s in the network slice request q_k residing on the physical node $p_i \in P$.
- With the decision variable $\gamma_{ij}^{st,k}$ taken as 1, the traffic between the virtual link $e_{st} \in e$ is routed through the physical path $\mathcal{L}_{ij}$ regarding q_k, and 0 otherwise. A single path is enforced to be unsplit since $\gamma_{ij}^{st,k}$ is an integer.

- $\zeta_i \in \{0,1\}$ denotes the occupancy status of p_i, with $\zeta_i = 1$, p_i hosts one or more VNFs; with $\zeta_i = 0$, p_i is available. $\zeta_i = 1 - \prod_{k\in K}[\prod_{v_s \in V}(1-\rho_i^{s,k})]$, here, for notational convenience, we introduce an auxiliary variable $\tau_i = \prod_{k\in K}[\prod_{v_s \in V}(1-\rho_i^{s,k})]$. $\zeta_i = 1 - \tau_i$ equals to 1, if more than one VNFs are resided on p_i, and 0 otherwise.

Objectives:

- Network slices are required to provide advanced services for the connected latency-sensitive devices, especially in the uRLLC use cases. The design objective for minimizing the transmission latency of slices is represented as follows:

$$min \quad f_u = \sum_{ij}\sum_{st}\{\gamma_{ij}^{st,k} \cdot \delta(\mathcal{L}_{ij}) + h(\mathcal{L}_{ij})\}, \forall k \in \mathcal{K}. \tag{3.2}$$

- Network slices need enhanced and sufficient computing resources to provide high data rates and better user experiences to customers, particularly in the eMBB use cases. The design objective for maximizing remaining computing resources of the physical network is defined as below:

$$max \quad f_e = \sum_{i}\sum_{s}\{\zeta_i(c_i + m_i) - \rho_i^{s,k}(c_s + m_s)\}, \forall k \in \mathcal{K}. \tag{3.3}$$

- Network slices are expected to support extremely high connection density of wireless devices when they transmit data simultaneously through the network, especially in the mMTC use cases. The minimum bandwidth utilization of occupied links is crucial for providing sufficient bandwidth capacity, the design objective is as follows:

$$min \quad f_m = \sum_{ij}\sum_{st}\{\gamma_{ij}^{st,k} \cdot b(e_{st}^k) \cdot h(\mathcal{L}_{ij})\}, \forall k \in \mathcal{K}. \tag{3.4}$$

Capacity Constrains:

$$\forall k \in \mathcal{K}, p_i \in P : \begin{cases} \sum_s \rho_i^{s,k} \cdot c_s \leq c_i \\ \sum_s \rho_i^{s,k} \cdot m_s \leq m_i \end{cases} \tag{3.5}$$

$$\sum_k \sum_{e_{st} \in e} \gamma_{ij}^{st,k} \cdot b(e_{st}^k) \leq b(\mathcal{L}_{ij}) \quad \mathcal{L}_{ij} \in L \tag{3.6}$$

Connectivity Constrains:

$$\begin{aligned} &\forall e_{st}^k \in E,\ s < t,\ \forall i,\ \forall k \in \mathcal{K} : \\ &\sum_{ij} [\gamma_{ij}^{st,k} - \gamma_{ji}^{st,k}] = \rho_i^{s,k} - \rho_i^{t,k} \end{aligned} \tag{3.7}$$

Variable Constraints:

$$\forall p_i \in P,\ \forall v_s^k \in V,\ \forall k \in \mathcal{K} \begin{cases} \sum_s \rho_i^{s,k} \leq 1 \\ \sum_i \rho_i^{s,k} = 1 \end{cases} \tag{3.8}$$

$$\forall p_i \in P,\ \forall v_s^k \in V,\ \forall k \in \mathcal{K} \begin{cases} \tau_i \leq 1 - \rho_i^{s,k} \\ \tau_i \geq 1 - \sum_s \rho_i^{s,k} \end{cases} \tag{3.9}$$

$$\zeta_i = 1 - \tau_i \quad \forall p_i \in P \tag{3.10}$$

$$\rho_i^{s,k},\ \gamma_{ji}^{st,k},\ \zeta_i,\ \tau_i = \{0,1\} \tag{3.11}$$

3.5 Algorithms of Basic Design Policy

In this section, the details of the proposed algorithm for the basic NSDP are provided to achieve a trade-off between computational complexity and performance.

The position of particle η is represented by $X_\eta^k = (x_{\eta,1}^k, x_{\eta,2}^k, .., x_{\eta,\varepsilon}^k, ..x_{\eta,z}^k)$, which can be considered as the design solution of VNFs for the slice request q_k, where ε denotes the VNF serial number and z denotes the total number of VNFs for q_k. The location number of the physical node denoted $x_{\eta,\varepsilon}^k$ can be represented as the design result of ε^{th} VNF in q_k. That is to say, the mapping location of ε^{th} VNF is $x^k_{\eta,\varepsilon}$ in the physical network. Specifically, the quality of X^k_η can be determined by the fitness function $F(q_k)$, which is defined as follows:

$$min \quad F(q_k) = w_u \cdot f'^k_u + w_m \cdot f'^k_m - w_e \cdot f'^k_e \tag{3.12}$$

where the normalization of f'^k_u, f'^k_m and f'^k_e are expressed as follows:

$$f'^k_u = \frac{f^k_u - min(f^k_u)}{max(f^k_u) - min(f^k_u) + 1} \tag{3.13}$$

$$f'^k_m = \frac{f^k_m - min(f^k_m)}{max(f^k_m) - min(f^k_m) + 1} \tag{3.14}$$

$$f'^k_e = \frac{f^k_e}{\sum_i \zeta_i (c_i + m_i)} \tag{3.15}$$

In the model, the values of f'^k_u, f'^k_m and f'^k_e will be set to infinity when the particles do not satisfy their capacity constraints. In addition, the weights w_u, w_m and w_e are denoted by the ratios of the numbers of uRLLC, mMTC and eMBB slices to the total number of NS requests, respectively.

During the design processes of network slices, each particle will deposit its previous positions after each iteration. The iterative updates of particles are performed in terms of their velocity and position, which are determined by three factors: the current position $x_{\eta,d}$, the best position of each particle $\mathcal{B}_{\eta,d}$ and the best global position of all particles $\mathcal{B}_{g,d}$. The velocity and position of the particle η during its iteration can be updated as follows:

$$\begin{aligned} \mathcal{V}_{\eta,d}(t+1) = \omega \mathcal{V}_{\eta,d}(t) + C_1 r_1 (\mathcal{B}_{\eta,d} - x_{\eta,d}(t)) + \\ C_2 r_2 (\mathcal{B}_{g,d} - x_{\eta,d}(t)) \end{aligned} \tag{3.16}$$

$$x_{\eta,d}(t+1) = x_{\eta,d}(t) + \mathcal{V}_{\eta,d}(t+1) \tag{3.17}$$

where $\eta = (1, 2, ..., \mathcal{M})$, $d = (1, 2, ..., z)$ and ω denotes the inertia weight. C_1 and C_2 are learning factors for adjusting the convergence of the algorithm, r_1 and r_2 stand for two random numbers which are uniformly distributed in [0,1].

Algorithm 3 Basic Network Slicing Design Policy Algorithm

```
Input:
    $G$: the physical network;
    $q_k$: the set of network slice requests ($k = 1, 2..., \mathcal{K}$);
Output:
    $\mathcal{D}$: the network slice design solution;
 1: for each network slice request $q_k$ do
 2:     construct a candidate position list for each VNF;
 3:     initialize position $X$ and velocity $\mathcal{V}$ for $\mathcal{M}$ particles;
 4:     for each iteration $\mathcal{N}_n$ do
 5:         for each particle $\eta$ do
 6:             if the particle $\eta$ satisfies the constrains (3.7)-(3.11) then
 7:                 calculate $F(q_k)$ based on the formula (3.12);
 8:             else
 9:                 make $f_u'^k$, $f_m'^k$ and $f_e'^k$ infinity;
10:             end if
11:             calculate $\mathcal{B}_{\eta,d}$ and $\mathcal{B}_{g,d}$;
12:             calculate $\mathcal{V}_{\eta,d}$, $x_{\eta,d}$ based on formulas (3.16), (3.17);
13:         end for
14:         if $\mathcal{N}_n$=$\mathcal{N}$ then
15:             go to step 18;
16:         else
17:             go back to step 5;
18:         end if
19:         return $\mathcal{D}_V^k$
20:     end for
21:     for each link $e_s \in E$ do
22:         map links based on the Dijkstra algorithm;
23:         return $\mathcal{D}_E^k$
24:     end for
25: end for
26: return $\mathcal{D}$
```

In this work, the matching degree $g(v_s, p_i)$ between each VNF $v_s \in v$ and the physical node p_i $(i = 1, 2.., |P|)$ is defined to initialize the position of particles by means of the formal contexts of the physical nodes and network slice requests. To be specific, the definition of $g(v_s, p_i)$ is shown as:

$$g(v_s, p_i) = \frac{|c_s \leq c_i| + |m_s \leq m_i|}{|A_{p_i} \cup I_v|} \tag{3.18}$$

where $A_{p_i} \cup I_v$ is the union set of resource attributes of p_i and v_s, and p_i can be the candidate position of v_s when $g(v_s, p_i) \geq 0.5$. The details of the basic network slicing design policy algorithm are shown in Algorithm 3, which implements the design processes of network slice requests by a set of efficient iterations. Besides, the time complexity of Algorithm 3 is $O(\mathcal{N} \cdot \mathcal{M} + |E|)$ for the design process of each network slice request.

3.6 Performance Evaluation

In this section, the experimental environment settings are introduced, and the main performance evaluations compared with several existing algorithms are presented.

3.6.1 Simulation Setup

In the simulation works, the physical network topology is randomly generated with 50 nodes and 130 links employing the GT-ITM tool [16], which is correspond to a medium-sized network. The CPU and memory capacity levels of physical nodes are uniformly distributed between 0 to 25 and the physical bandwidth resource capacities have uniform distribution of $U[0, 100]$.

In this work, the design process of network slice requests is online. It is assumed that each network slice request arrives according to a Poisson Process, and each of them has an exponential distributed Time-To-Live (TTL) with an average of $\mu = 1000$ time units. In the network topology of each slice, the number of VNFs is randomly determined a uniform distribution of $U[2, 10]$ following similar setups to previous work [9]. Besides, the CPU and memory requirements of each virtual node obey to the uniform distribution of $U[0, 10]$ and the bandwidth requirements are uniformly distributed between 0 to 50.

The number of particles $\mathcal{M}$ is set to 5 and the threshold of iteration $\mathcal{N}$ is set to 50. The initial values of ω, C_1 and C_2 are set to 0.729, 2 and 2 respectively, which

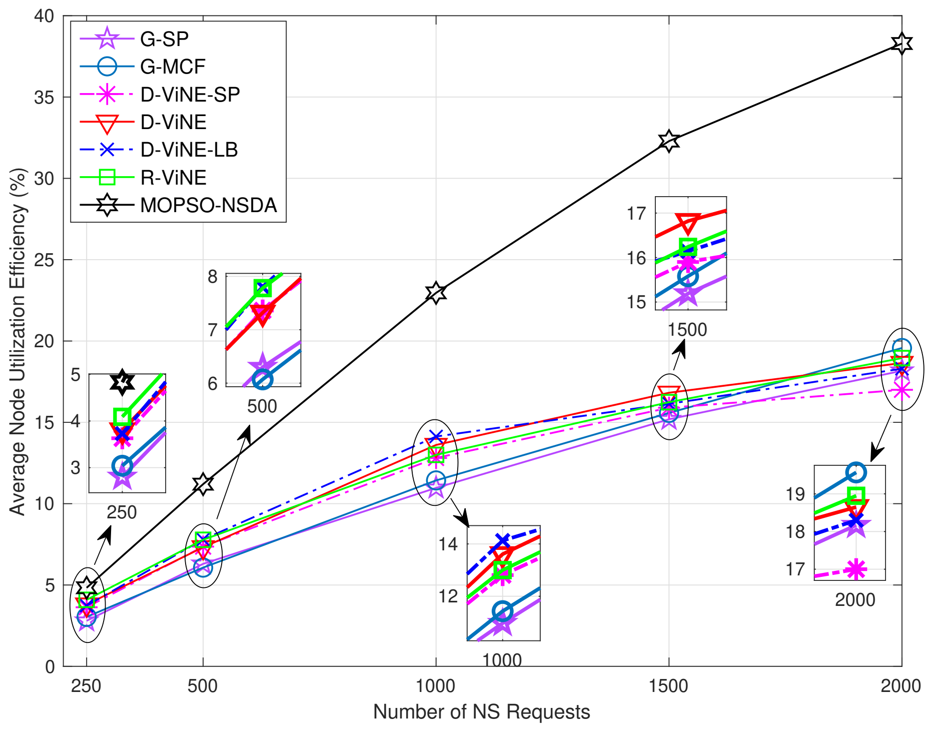


Fig. 3.3 Average Physical Node Utilization Efficiency

are set based on the experimental results of the Reference [101]. The weights w_u, w_e and w_m are set as the same value $1/3$, which means the numbers of different kinds of slices are the same in our simulations. Moreover, in our evaluations, we compare our proposed approach MOPSO-NSDA with several classic algorithms, such as G-MCF[9], G-SP[30], R-ViNE, D-ViNE, D-ViNE-SP and D-ViNE-LB [31]. In addition, 250, 500, 1000, 1500 and 2000 network slice requests are set to evaluate the algorithms in this chapter. The algorithms were set to run until 50,000 time units. And all simulations are carried out on a laptop with eight 1.9GHz CPU cores and 16GB memories.

3.6.2 Evaluations Results

Different performance metrics for evaluation purposes are employed in our simulation, such as the average node and link utilization efficiencies, the node and link occupancy ratios and the execution time of the proposed algorithm.

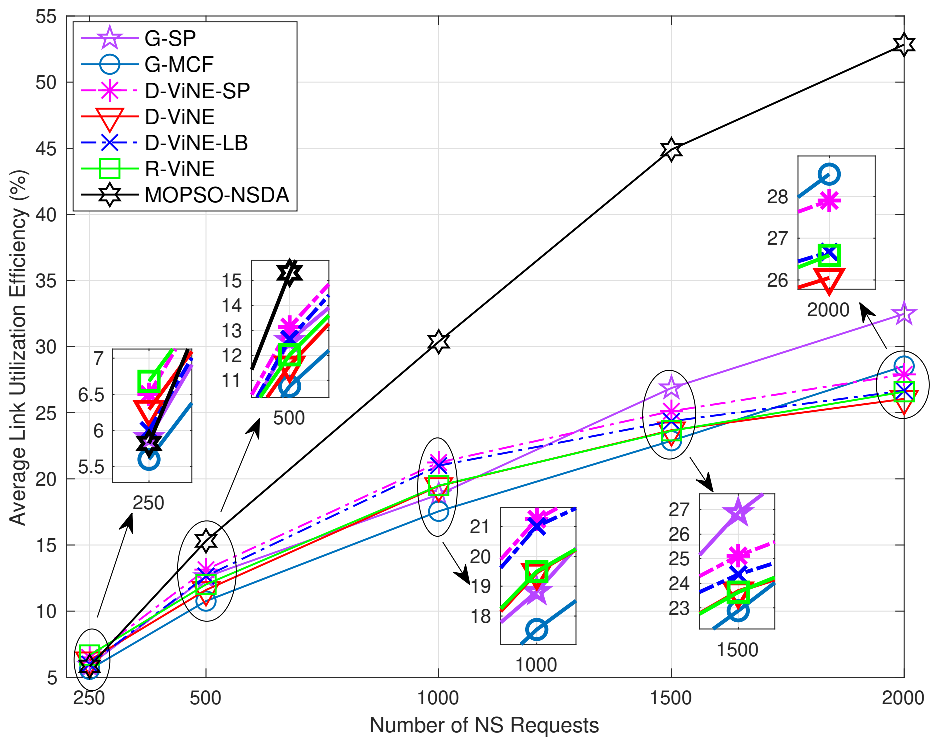


Fig. 3.4 Average Physical Link Utilization Efficiency

Fig. 3.3 and Fig. 3.4 demonstrate the average node and link utilization efficiencies, the proposed algorithm leads to better results than the existing comparison algorithms due to the help of the local and global optimal mapping positions of particles during the iterations. Moreover, the different kinds of resource capacities are considered for various use cases, such as transmission latency, computing resources and bandwidth capacity during the slice design processes. The G-SP is the baseline algorithm that adopts the greedy strategy node mapping with k-shortest link mapping algorithms. From the results, it can be seen that the average node and link resource utilization efficiencies are 38.29% and 52.84% for 2000 network slice requests of the proposed algorithm, respectively, which means that they can obtain up to twice of that compared with the G-SP and other algorithms. In addition, the highest node and link resource utilization efficiencies of the proposed algorithm imply its highest acceptance ratio.

From Fig. 3.5 and Fig. 3.6, it is evident that the node and link occupancy ratios gradually increase as the number of NS requests is increased. Regarding different time

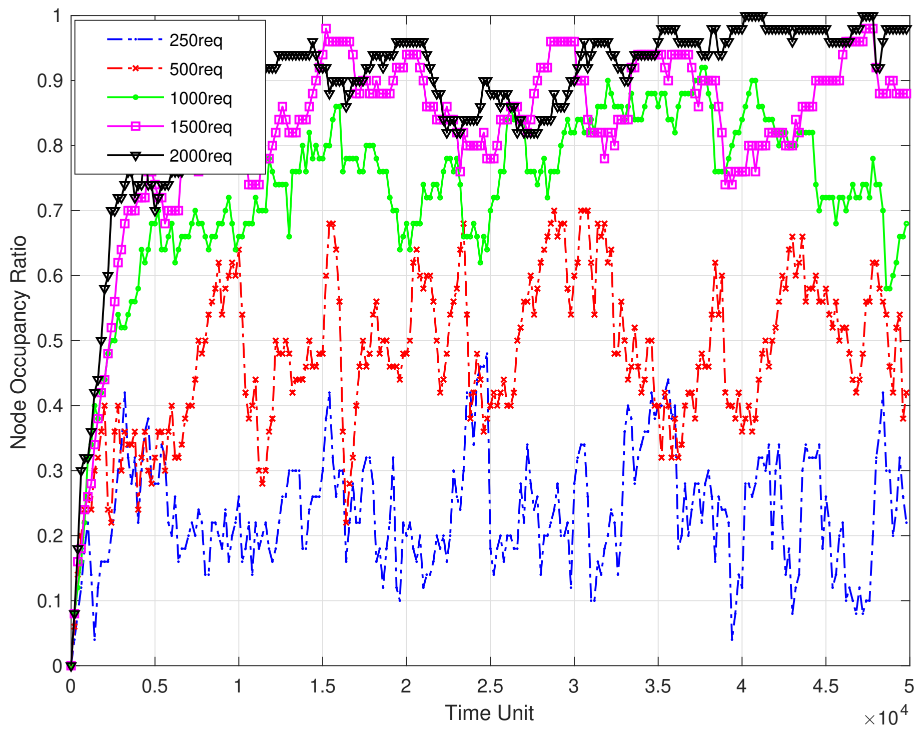


Fig. 3.5 Physical Nodes Occupancy Ratio

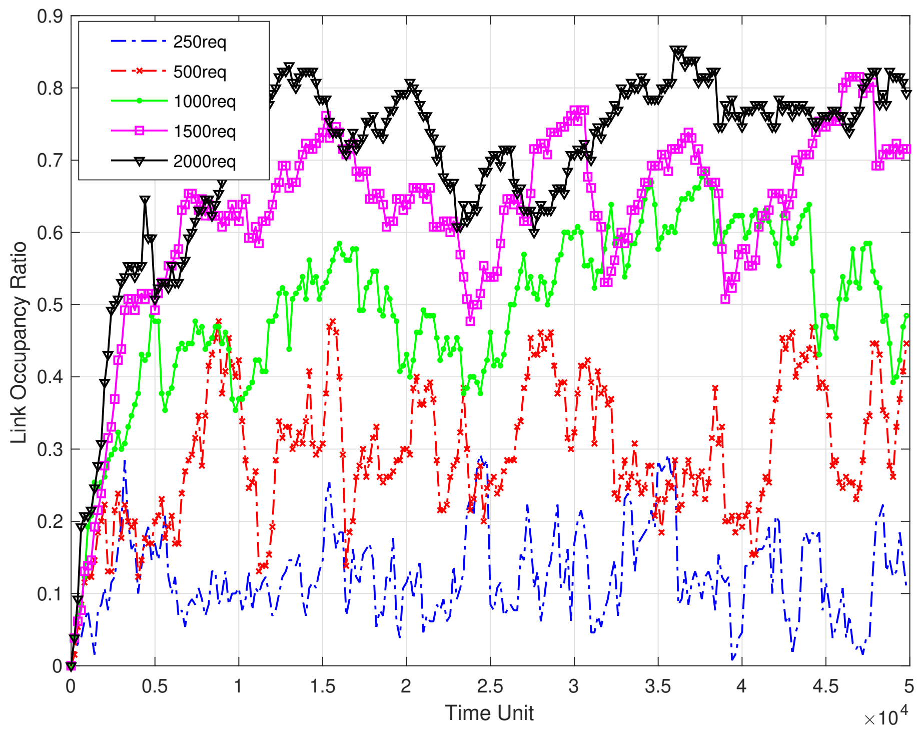


Fig. 3.6 Physical Links Occupancy Ratio

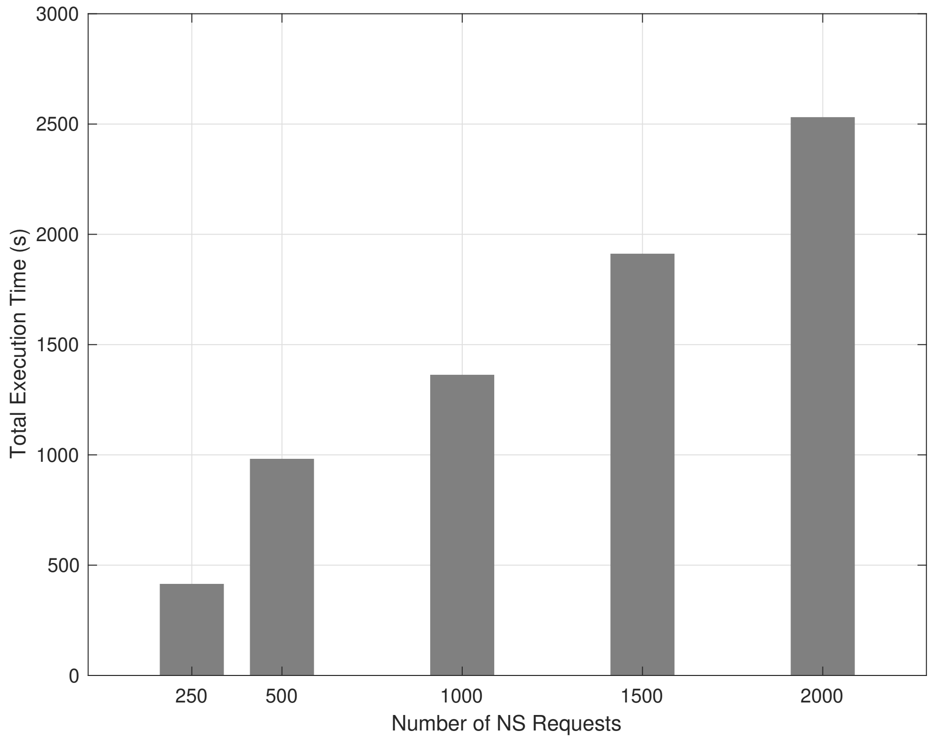


Fig. 3.7 Execution Time

units, the occupancy ratios of the proposed algorithm fluctuate within a certain range, which implies that it can guarantee the balanced utilization of physical nodes and links of the system. However, the tendencies of different numbers of network slice requests in terms of time unit are still fluctuated, because different slice requests have different TTL so that will can be removed from the physical network when they are expired.

Furthermore, the total execution time of the proposed algorithm in terms of different number of network slice requests is depicted in Fig. 3.7. It is obvious that the more network slicing request is, the longer execution time is. From the simulation results, the average execution time for each network slice request design process is around $1.5006s$.

3.7 Conclusions

Network slicing is recognized as a promising technology, which can not only allow the physical network to divide into different network slices but provide various service for different application scenarios. For the implementation of network slicing, the design policy is regarded as one of the most significant issues. Thus, this chapter presents an ILP formulation and a meta-heuristic algorithm for the basic NSDP problem. To be specific, the simulations results show that the performance of resource utilization efficiency of the proposed algorithm is better than other existing algorithms. The analysis of average node and link occupancy ratios illustrates that the proposed algorithm can guarantee the balanced utilization of system resource. And the analysis shows that the execution time of network slice design process is fast with the proposed algorithm. Further, a service-aware design policy will be investigated in Chapter 4 based on the present basic design policy.

Chapter 4

Service-aware Design Policy of End-to-End Network Slicing For 5G Use Cases

4.1 Introduction

Recently, the NSDP is being investigated extensively in both academia and industries. Some uncertain scenarios are considered in the network slicing research, because the traffic demands in different slices may fluctuate or they may burst in a period of time.

In existing works, the robust optimization methodology is regarded as a conducive approach to solve the uncertain problems. It contains no assumption that the probability distributions should be available in advance, leading to computationally tractable formulations [57]. For instance, Bertsimas and Sim [102] developed a Γ-robust model to deal with the data uncertainty for network flows and discrete optimization, which can be deemed as a main foundation stone to deal with uncertain demands in networks.

On this basis, some related works are proposed, for example, a Γ-robust optimization model was presented to solve the virtual network embedding problem for the networks of large scale with uncertain demands in Reference [103]. Also, Reference [104] employed the Γ-robust uncertainty set proposed in [102] to handle traffic uncertainties by presenting two optimization models, which aimed to reduce the deployment cost of slices. And a study about robust network slicing schemes was investigated in Reference [105] to deal with the recovery and reconfiguration of slices in a unified framework, to minimize the bandwidth consumption. Reference [106] illustrated a solution of the communication network planning based on the robust optimization using an accurate mathematical model with demand uncertainty.

The formulations of the NSDP considering the uncertainty in traffic demands were proposed in References [104, 103]. However, the differences among the characteristics of diverse slice use cases are neglected. In addition, a service-focused deployment algorithm of network slicing [95] was proposed to the realize efficient utilization of resources for different slices. Nevertheless, the traffic demands in diverse slice requests are ignored. In practice, network slices usually have different performance requirements for various service application scenarios. The traffic demands in network slices may fluctuate to some extent. Thus, it is significant to explore the service-aware NSDP for various use cases considering fluctuated traffic demands.

This chapter aims to propose a versatile service-aware network slice design policy, which is appropriate for the application scenario that different use cases can coexist in an efficient way, whilst considering the fluctuation of traffic demands. The present design policy can not only enhance resource efficiency but also guarantee the various service requirements.

Firstly, multiple design objectives for eMBB, mMTC and uRLLC use cases are introduced, respectively. Then, two optimization models of the service-aware NSDP are formulated, including the deterministic model and the robust model. Similar to the basic NSDP problem, the formulated service-aware NSDP is also an Integer Linear Programming problem, which is a NP-Hard problem. Besides, the NSDP problem evolves from the VNE problems, which means that it may be hard to solve the problem within a polynomial computing time when its input size is considerably large [9]. Thus, a heuristic algorithm is proposed, called the service-aware network slicing design policy (S-NSDP) algorithm. It can be implemented via efficient iterations, which is inspired by the Multi-Objective Particle Swarm Optimization (MOPSO) algorithm [55]. The contributions of this chapter can be summarized as follows:

- Multiple design objectives for eMBB, mMTC and uRLLC use cases are proposed, respectively, concerning different priorities, including bandwidth resource utilization, CPU and memory resource utilization and latency, providing various services and satisfying performance requirements in different 5G application scenarios.
- Two optimization models of the NSDP are formulated: first, the deterministic formulation is considered as the nominal case with nominal traffic demands; then, the robust formulation is adopted as an extended version of the deterministic formulation to deal with uncertain traffic demands, where equivalent robust counterparts can be obtained with robust coefficients Γ and Ψ.
- The S-NSDP algorithm is proposed with fluctuated traffic demands, which can not only utilize resources efficiently but also ensure the different performance requirements of slices. It can be considered as a trade-off between computational effort and the quality of solutions. Specifically, the initialization algorithm of particle swarms and the update scheme of particle positions are presented.
- Extensive simulations are carried out to validate the present design policy, the impact on objective values is analyzed considering the robustness optimization. And the algorithm performance is evaluated compared with existing work. A comprehensive enhancement is demonstrated by the proposed algorithms, of resource utilization, resource efficiency and acceptance ratio.

The rest of this chapter is organized as follows: Section 4.2 provides a network model of the service-aware NSDP. Section 4.3 presents the details of multiple service-aware design objectives for different slices, and the optimization models of the NSDP, including the deterministic model and the robust model, with resource constraints. Besides, the proposed algorithms of the service-aware network slicing design policy are introduced in Section 4.4 and evaluated in Section 4.5 with a detailed analysis. Finally, a brief conclusion is presented in Section 4.6.

4.2 Network Model of Service-aware Network Slicing Design Problem

The basic mathematical models of the physical infrastructure and network slice requests have been introduced in Section 3.2. In this section, the additional information on the network models related to the service-aware NSDP is supplemented. The notations used to define the NSDP problem are summarized in Table 4.1.

4.2.1 Physical Infrastructure Model

The physical infrastructure is associated with an undirected graph $G = (P, L, A_P, A_L)$. $A_{p_i} \in A_P$ is defined by $A_{p_i} = (\{c_{p_i}, m_{p_i}, \Delta_{p_i}\} | p_i \in P)$, where c_{p_i}, m_{p_i} and Δ_{p_i} represent the CPU capacity, memory capacity and node type of p_i, respectively. Δ_{p_i} is a binary parameter, and $\Delta_{p_i} = 0$ if p_i is an access node. Besides, $A_L = (\{b(\mathcal{L}_{ij}), h(\mathcal{L}_{ij})\}$, where $b(\mathcal{L}_{ij})$ represents the bandwidth capacity of the physical path $\mathcal{L}_{ij}$, and $h(\mathcal{L}_{ij})$ indicates its total latency.

4.2.2 Network Slice Request Model

A network slice request q_k is represented by $q_k = (v, e, o, \mathcal{T}_k, D_k)$, where $v \in V$, $e \in E$. o denotes its Time-To-Live (TTL), which indicates that q_k will be removed when its TTL expires. And $\mathcal{T}_k$ denotes the type of use case of q_k.

Given a set of end-to-end traffic demands D_k for q_k, which is defined by $D_k = (v_s, v_t, d_k | \forall k \in \mathcal{K})$. Each traffic d_k is routed from its source node $v_s \in v$ towards its destination node $v_t \in v$ through the virtual link $e_{st} \in e$. D_k leaves sources and arrives at their destinations by routing through a collection of VNFs in a predefined order. The set of sources can be represented by $S(D_k) = \{(v_s, v_t) = e_{st} \in e(d_k) | v_s = s(d_k), \forall d_k \in D_k\}$, similarly, the set of destinations is denoted by $H(D_k) = \{(v_s, v_t) = e_{st} \in e(d_k) | v_t = h(d_k), \forall d_k \in D_k\}$.

I_{v_s} denotes the required attributes of the VNF v_s, which can be represented by $I_{v_s} = (\{c_{v_s}, m_{v_s}, \varepsilon_{v_s}, c_{\lambda_{v_s}}, m_{\lambda_{v_s}}\} | v_s \in v)$. Specifically, c_{v_s} and m_{v_s} specify the size of CPU and memory resource blocks required by the VNF v_s for its implementation in the

Table 4.1 Notations of the service-aware NSDP

Notation	Description
G	Physical infrastructure topology
P	Set of physical nodes
L	Set of physical links
$\mathbb{T}$	Set of relations of physical nodes and their attributes
A_P	Set of attributes of P
A_L	Set of attributes of L
Q	Network slice topology
V	Set of virtual network functions
E	Set of virtual links
D_k	Set of traffic demands of q_k
$S(D_k)$	Set of sources of D_k
$H(D_k)$	Set of destination of D_k
q_k	A slice request
$\mathcal{L}_{ij}$	Physical path between the physical nodes p_i and p_j
e_{st}	Virtual link between the VNFs v_s and v_t
d_k	A traffic demand in D_k
c_{p_i}	CPU capacity of p_i
m_{p_i}	Memory capacity of p_i
$b(\mathcal{L}_{ij})$	Bandwidth capacity of $\mathcal{L}_{ij}$
$h(\mathcal{L}_{ij})$	Latency of $\mathcal{L}_{ij}$
I_{v_s}	Set of required attributes of v_s
c_{v_s}	Size of CPU resource blocks required by v_s
m_{v_s}	Size of memory resource blocks required by v_s
ε_{v_s}	Elastic coefficient of v_s
$c_{\lambda_{v_s}}$	CPU resource coefficient of v_s
$m_{\lambda_{v_s}}$	Memory resource coefficient of v_s
$\rho_i^{s,k}$	Decision variable of VNFs
$\gamma_{ij}^{st,k}$	Decision variable of virtual links
ζ_i	Variable of occupancy status of p_i
x_{v_s}	Number of resource blocks required by v_s

infrastructure. Besides, $c_{\lambda_{v_s}}$ denotes the CPU resource coefficient of v_s, meaning that it requires $c_{\lambda_{v_s}}$ ($c_{\lambda_{v_s}} > 0, \forall v_s \in v$) CPU resources to process one unit of traffic demand, similarly to the memory resource coefficient $m_{\lambda_{v_s}} > 0$. Let ε_{v_s} represent the elastic coefficient of v_s, which indicates that the traffic demand d_k may fluctuate to $d_k * \varepsilon_{v_s}$ ($\varepsilon_{v_s} \geqslant 0, \forall v_s \in v$) after it routes through v_s.

4.3 Problem Formulation

4.3.1 Service-aware Design Objectives

In this chapter, it is aimed to propose a versatile service-aware network slice design policy to support various application scenarios. Thus, a comprehensive design objective function is necessary. For different 5G use cases, such as eMBB, uRLLC and mMTC, they are served by a set of network slices. The network slices belonging to different use cases should satisfy their specific service requirements, such as latency, computing and bandwidth resources. Each use case has its own priority of service requirements, which should be fully considered.

In detail, it is essential to ensure that uRLLC slices have a high priority in end-to-end latency. They need extremely low latency and significantly high availability to promise high-quality services. For instance, they are expected to deliver advanced services for connected latency-sensitive devices, such as intelligent transportation vehicles, remote health-care tools, tactile communications and virtual reality.

In the present work, the latency $h(\mathcal{L}_{ij})$ of the physical path $\mathcal{L}_{ij}$ between p_i and p_j is determined by its transmission delay, propagation delay and the processing delay of VNFs and physical nodes. The maximum transmission delay of $\mathcal{L}_{ij}$ is denoted by ξ_{ij} when its all bandwidth capacities are occupied. Further, the transmission delay of $\mathcal{L}_{ij}$ indicates as $\mu_{ij} \cdot \xi_{ij}$, where μ_{ij} is represented by the utilization ratio of $\mathcal{L}_{ij}$ and it can be expressed as:

$$\mu_{ij} = \frac{\sum_{k \in K} \sum_{e_{st} \in e} \gamma_{ij}^{st,k} \varepsilon_{v_s} d_k}{b(\mathcal{L}_{ij})}, \tag{4.1}$$

where $\gamma_{ij}^{st,k}$ denotes the decision variable, with it taken as 1, the traffic between the virtual link $e_{st} \in e$ is routed through the physical path $\mathcal{L}_{ij}$ regarding the traffic $d_k \in D_k$; and 0 otherwise.

Let κ_{ij} denote the propagation latency of $\mathcal{L}_{ij}$, it is defined by the number of hops between $\mathcal{L}_{ij}$. The processing latency of p_i is indicated by $\iota_{p_i}(\forall p_i \in P)$. And $\iota_{v_s}(\forall v_s \in V)$ denotes the processing latency of v_s. The total latency δ_{ij} of $\mathcal{L}_{ij}$ increases as more VNFs are placed on p_i and p_j. The network slice design objective regarding latency is shown as:

$$max \quad f_u = \mathcal{C} - \sum_{\mathcal{L}_{ij}} \{ \frac{\sum_{k \in \mathcal{K}} \sum_{e_{st} \in e} \gamma_{ij}^{st,k} \varepsilon_{v_s} d_k}{b(\mathcal{L}_{ij})} \xi_{ij} + \kappa_{ij} + \sum_{k \in \mathcal{K}} \sum_{p_i \in \mathcal{L}_{ij}} (\iota_{p_i} + \sum_{v_s \in v} \rho_i^{s,k} \iota_{v_s}) \}, \tag{4.2}$$

where $\mathcal{C}$ is a sufficiently large constant. $\rho_i^{s,k}$ indicates the decision variable for determining the traffic demand d_k to route through the VNF v_s placing onto the physical node p_i.

The network slicing technique facilitates the mobile network operators to provide improved user experience and high data rate services by creating different slice service instances. Under these circumstances, massive traffic data will be transmitted simultaneously through networks. Therefore, the effective use of bandwidth resources should be guaranteed in the design of network slicing.

Particularly, it is significant to satisfy the requirements of bandwidth resources for eMBB slices, which can prevent the exhaustion of bandwidth resources from affecting their service qualities. Because eMBB slices usually require high traffic demands, such as ultra HD video stream, cloud storage and mobile augmented reality. Specifically, the slice design objective considering bandwidth resources is expressed as:

$$max \quad f_e = \sum_{k \in \mathcal{K}} \sum_{e_{st}(v_s) \in e} \sum_{\mathcal{L}_{ij}} \{ \gamma_{ij}^{st,k} \varepsilon_{v_s} d_k \}. \tag{4.3}$$

In addition, sufficient computing resources, such as CPU and memory resources, are crucial for the implementation of network slicing. For instance, in IoT application scenarios of mMTC use cases, a growing number of intelligent devices are enabled for consumer use, including wearable devices, automation systems and some appliances with remote monitoring capabilities. Thus, it is important to supply enhanced and abundant computing resources for the services of the high density of diverse connections preferentially. The network slice design objective related to computing resources is defined by:

$$max \quad f_m = \sum_{k \in \mathcal{K}} \sum_{p_i \in P} \sum_{v_s \in v} \rho_i^{s,k} \{x_{v_s}(c_{v_s} + m_{v_s}) + (c_{\lambda_{v_s}} + m_{\lambda_{v_s}})\varepsilon_{v_s} d_k\}, \tag{4.4}$$

where $x_{v_s} \in \mathbb{R}_{\geqslant 0}$ specifies the number of resource blocks required by v_s.

In conclusion, the intention in setting out the multiple design objectives is to propose a comprehensive design objective function. The proposed design policy is appropriate for various application scenarios, where tenants can require different types of network slices in the same infrastructure. Therefore, the design objective function of the service-aware NSDP is formulated by using the Simple Additive Weighting (SAW) method, which is shown as:

$$max \quad \omega_u f_u + \omega_e f_e + \omega_m f_m \tag{4.5}$$

where the weighting coefficients ω_u, ω_e and ω_m are defined as the degrees of priority regarding different types of network slices. They are respectively denoted by the ratios of the number of uRLLC, eMBB and mMTC network slice requests to the total size of the network slice set.

In fact, the SAW method and the ε- constraint model both are nominal models for solving Multi-objective Optimization Problems (MOPs). Both techniques aim to reduce the dimensionality of the multiple objectives in a certain way for MOP problems. In essence, both two methods transform a MOP problem into a Single-objective Optimization Problem (SOP), which make it easier to be solved. In addition to the SAW method, the ε- constraint method also seems to work to solve the NSDP.

However, the SAW method is considered as a more appropriate way to solve the proposed NSDP problem compared to the ε- constraint model. First of all, this chapter aims to propose a versatile network slice design policy to guarantee the different services in various use cases, which does not focus on one specific slice scenario like other existing works [107], [108]. Besides, different types of network slices belonging to diverse use cases are accommodated in the same underlying physical infrastructure, and the concordant coexistence of various slices should be guaranteed. Hence, a general design objective function is necessary to be proposed for various eMBB, uRLLC and mMTC network slices.

If the ε- constraint model is adopted in the proposed NSDP problem, a specific design objective should be selected from the k objectives as the main objective, and the other $k-1$ objectives will be considered as constraints by adding different thresholds. Therefore, it is not suitable for the NSDP problem since it would be difficult to select a function as the primary objective function, especially considering different network slices.

To put it differently, the ε- constraint model is more suitable to solve the NSDP problem for a specific use case rather than a general NSDP problem. Because it is pertinent to select the main objective in a specific NSDP problem, so that it can be better optimized. For instance, the latency objective can be determined as the main objective and the computing resource and bandwidth resource objectives can be considered as constraints in the NSDP problem for the uRLLC use case. However, for the service-are NSDP problem proposed in this chapter, it should avoid selecting a function as the main objective. Because it is infeasible to choose any objective as the main objective for a general problem and the diverse use cases have different priorities of service requirements.

Besides, the proposed NSDP problem can be solved and implemented more efficiently by using the SAW method than the ε- constraint model. If the ε- constraint method is employed, an appropriate way to solve the proposed NSDP problem is to transform it into three different specific problems. Specifically, three different optimization models should be formulated with different selected main objectives. The value of threshold $\varepsilon_i(i = 1,2,3...)$ needs to be set differently in various NSDP scenarios, and then the optimization problems will be solved three times separately for eMBB, uRLLC and mMTC use cases. Moreover, it will have a certain impact on the optimal solutions whether ε_i is set appropriately or not. New constraint variables ε_i are introduced, increasing the number of constraints in the optimization problem, which will cause that the difficulty of solving the specific NSDP problems tends to increase.

In addition, the unique solution of the proposed NSDP problem is Pareto optimal by using the SAW method when the weighting coefficients are defined as $\omega_u, \omega_e, \omega_m \geq 0$. And it is strictly positive for at least one design objective, and $\omega_u + \omega_e + \omega_m = 1$. The corresponding proofs can be found in Formula (16) and Theorem 4 in Reference [52].

Particularly, different application scenarios of network slices can be realized at ease by adjusting ω_u, ω_e and ω_m. For instance, for the scenario containing three different kinds of slice requests with the same priority, where $\omega_u = \omega_e = \omega_m = 1/3$. In the following, the analysis of the different scenarios is given in Section 4.5, with different values of ω_u, ω_e and ω_m.

4.3.2 Deterministic Formulation

In this subsection, a deterministic optimization model of the service-aware NSDP is formulated, which aims to enhance the resource utilization efficiency for hosting as many network slice requests as possible in the network while satisfying their different service requirements. The proposed model places VNFs and routes virtual links in the physical infrastructure. It is assumed that the traffic demands in different slice requests are deterministic. That is to say, the volume of traffic demand is fixed after the creation of a network slice request. The deterministic problem of the service-aware NSDP is formulated as follows:

$$max \quad g_d = \omega_u f_u + \omega_e f_e + \omega_m f_m \tag{4.6.1}$$

$$\textbf{s.t.} \quad \sum_{k \in \mathcal{K}} \sum_{v_s \in v} \rho_i^{s,k} (x_{v_s} c_{v_s} + c_{\lambda_{v_s}} \varepsilon_{v_s} d_k) \leq c_{p_i} \qquad \forall p_i \in P \tag{4.6.2}$$

$$\sum_{k \in \mathcal{K}} \sum_{v_s \in v} \rho_i^{s,k} (x_{v_s} m_{v_s} + m_{\lambda_{v_s}} \varepsilon_{v_s} d_k) \leq m_{p_i} \qquad \forall p_i \in P \tag{4.6.3}$$

$$\sum_{k \in \mathcal{K}} \sum_{e_{st}(v_s) \in e} \gamma_{ij}^{st,k} \varepsilon_{v_s} d_k \leq b(\mathcal{L}_{ij}) \quad \forall \mathcal{L}_{ij} \in L, \forall v_s \in e_{st} \tag{4.6.4}$$

$$\sum_{v_s \in s(d_k)} \gamma_{ij}^{st,k} - \sum_{v_s \in h(d_k)} \gamma_{ji}^{st,k}$$

$$= \begin{cases} 1_{v_s \in s(d_k)} - \rho_i^{t,k} & \forall k \in \mathcal{K}, e_{st} \in \mathcal{L}_{ij}(v_s)+, v_s \in v \\ \rho_i^{s,k} - \rho_i^{t,k} & \forall k \in \mathcal{K}, e_{st} \in e/\{s(d_k) \cup h(d_k)\} \\ \rho_i^{s,k} - 1_{v_s \in h(d_k)} & \forall k \in \mathcal{K}, e_{st} \in \mathcal{L}_{ij}(v_s)-, v_s \in v \end{cases} \tag{4.6.5}$$

$$\sum_{k \in \mathcal{K}} \sum_{p_i \in P} \rho_i^{s,k} = 1 \quad \forall v_s \in v \tag{4.6.6}$$

$$\begin{cases} \tau_i \leq \sum_{k \in \mathcal{K}} (1 - \rho_i^{s,k}) & \forall p_i \in P, \forall v_s \in v \\ \tau_i \geq 1 - \sum_{k \in \mathcal{K}} \sum_{v_s \in v} \rho_i^{s,k} & \forall p_i \in P \end{cases} \tag{4.6.7}$$

$$\zeta_i = 1 - \tau_i \quad \forall p_i \in P \tag{4.6.8}$$

$$\rho_s^i,\ \gamma_{ij}^{st,k},\ \zeta_i,\ \tau_i = \{0, 1\} \tag{4.6.9}$$

where constraints (4.6.2)-(4.6.3) ensure that the required resources for the implementation of slices and the processing of the traffic volume $d_k \in D_k$ should not exceed the system resource capacities. Constraint (4.6.4) guarantees that the bandwidth capacity of $\mathcal{L}_{ij}$ should be capable of supporting the traffic demands routing through e_{st} that are established on $\mathcal{L}_{ij}$. Constraint (4.6.5) enforces flow conservation at $p_i \in P$. Constraints (4.6.6)-(4.6.8) give the restriction of the occupancy status of $p_i \in P$.

4.3.3 Robust Formulation

In the previous subsection, the deterministic formulation of the NSDP is introduced. However, the assumption that the traffic demands are deterministic may not be appropriate in many practical application scenarios.

Actually, it is usually assumed that slice requests require resources according to the peak traffic demands to guarantee the services. But, it is not efficient since resources are unlikely to be fully utilized most of the time. Besides, it may cause infeasibility if traffic demands fluctuate, any increase in traffic demands can lead to the failed deployment of slice requests. Thus, a promising way to remedy these limitations is to consider traffic uncertainty in the formulation of the NSDP.

In the present work, the Robust Optimization (RO) methodology is applied, which has been widely adopted to deal with the challenges arising from uncertain situations [102]. A robust approach for the NSDP is proposed by combining the Ψ-robust and Γ-robust models. It is appropriate to the stochastic traffic demands because: 1) the number of uncertain traffic demands that can fluctuate simultaneously can be adjusted easily by changing the parameter Γ; 2) on basis of 1), the deviation of traffic demands can be regulated by the parameter Ψ. To capture this uncertainty, the uncertainty sets proposed by [109] is employed to formulate the uncertain traffic demands. The traffic demands are modeled as a 'box+polyhedral' uncertainty set U, which is the intersection between the polyhedral and box [58]. U is defined by:

$$U = \{\sigma_k | \|\sigma_k\|_\infty \leq \Psi, \|\sigma_k\|_1 \leq \Gamma\}. \tag{4.7}$$

Assume that the traffic demands fluctuate independently, which indicates there is no correlation among the elements in the uncertain set U. Each uncertain traffic demand $\tilde{d}_k (k \in K)$ is denoted by:

$$\tilde{d}_k = \bar{d}_k \pm \sigma_k \hat{d}_k, \tag{4.8}$$

where $\tilde{d}_k$ takes value in the symmetric interval $[\bar{d}_k - \sigma_k \hat{d}_k, \bar{d}_k + \sigma_k \hat{d}_k]$ with the nominal traffic demand $\bar{d}_k$. Besides, traffic demands can fluctuate by the positive perturbation $\hat{d}_k$ and $\sigma_k \in U$.

Assuming the number of traffic demands that deviate from their nominal values simultaneously is at most Γ, and the maximum deviation magnitude is Ψ. To immunize against infeasibility and obtain solutions that remain feasible for any independent random variables in the given uncertainty set U, resource constraints (4.6.2), (4.6.3) can be updated as their non-linear robust counterparts, which are shown as follows:

$$\begin{cases} \sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}c_{\lambda_{v_s}}\varepsilon_{v_s}\bar{d}_k+\sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}x_{v_s}c_{v_s}+ \\ \max_{\substack{J\in\mathcal{K},\\ |J|\leq\Gamma,|\sigma_k|\leq\Psi}}\{\sum_{k\in J}\sum_{v_s\in v}\rho_i^{s,k}c_{\lambda_{v_s}}\varepsilon_{v_s}\sigma_k\hat{d}_k\}\leq c_{p_i} \\ \\ \sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}m_{\lambda_{v_s}}\varepsilon_{v_s}\bar{d}_k+\sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}x_{v_s}m_{v_s}+ \\ \max_{\substack{J\in\mathcal{K},\\ |J|\leq\Gamma,|\sigma_k|\leq\Psi}}\{\sum_{k\in J}\sum_{v_s\in v}\rho_i^{s,k}m_{\lambda_{v_s}}\varepsilon_{v_s}\sigma_k\hat{d}_k\}\leq m_{p_i}. \end{cases} \tag{4.9}$$

Further, the non-linear robust constraints (4.9) can be linearized by abstracting them as exponential linear constraints. Each of them accounts for the scenario where a possibility of the given uncertainty set U is realized, which is modeled as:

$$\begin{cases} \sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}\{x_{v_s}c_{v_s}+c_{\lambda_{v_s}}\varepsilon_{v_s}(\bar{d}_k+\sigma_k\hat{d}_k)\}\leq c_{p_i} \\ \\ \sum_{k\in\mathcal{K}}\sum_{v_s\in v}\rho_i^{s,k}\{x_{v_s}m_{v_s}+m_{\lambda_{v_s}}\varepsilon_{v_s}(\bar{d}_k+\sigma_k\hat{d}_k)\}\leq m_{p_i}. \end{cases} \tag{4.10}$$

The robust NSDP with exponential constraints can be circumvented by employing the duality technique proposed by [102]. Its tractable reformulation can be modeled, where the inner maximization problems in the constraints (4.9) are converted to their corresponding dual equivalent:

$$\begin{cases} \max\limits_{\substack{J\in\mathcal{K},\\ |J|\le\Gamma,|\sigma_k|\le\Psi}} \sum\limits_{k\in J}\sum\limits_{v_s\in v} \rho_i^{s,k} c_{\lambda_{v_s}} \varepsilon_{v_s} \sigma_k \hat{d}_k = min \sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \Psi \varsigma_i^{s,k} + \Gamma z_i \\ \max\limits_{\substack{J\in\mathcal{K},\\ |J|\le\Gamma,|\sigma_k|\le\Psi}} \sum\limits_{k\in J}\sum\limits_{v_s\in v} \rho_i^{s,k} m_{\lambda_{v_s}} \varepsilon_{v_s} \sigma_k \hat{d}_k = min \sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \Psi \varsigma_i^{s,k} + \Gamma z_i \\ \varsigma_i^{s,k} + z_i \ge \hat{d}_k \rho_i^{s,k} \quad \forall k\in\mathcal{K} \\ \varsigma_i^{s,k} \ge 0, z_i \ge 0 \qquad \forall k\in\mathcal{K} \end{cases} \tag{4.11}$$

where $\varsigma_i^{s,k}$ and z_i are dual variables.

For fixed values of robust parameters $\Gamma\in\mathbb{Z}_{\ge 0}$ and $\Psi\in\mathbb{R}_{\ge 0}$, the robust counterparts to constraints (4.6.2) and (4.6.3) correspond to:

$$\begin{cases} \sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \rho_i^{s,k}(x_{v_s}c_{v_s} + c_{\lambda_{v_s}}\varepsilon_{v_s}\bar{d}_k) + \Psi\sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \varsigma_i^{s,k} + \Gamma z_i \le c_{p_i} \\ \sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \rho_i^{s,k}(x_{v_s}m_{v_s} + m_{\lambda_{v_s}}\varepsilon_{v_s}\bar{d}_k) + \Psi\sum\limits_{k\in\mathcal{K}}\sum\limits_{v_s\in v} \varsigma_i^{s,k} + \Gamma z_i \le m_{p_i} \\ \varsigma_i^{s,k} + z_i \ge \hat{d}_k \rho_i^{s,k} \\ \varsigma_i^{s,k} \ge 0, z_i \ge 0 \quad \forall v_s\in v, \forall p_i\in P, \forall k\in\mathcal{K} \end{cases} \tag{4.12}$$

Similarly, the constraint (4.6.4) can be updated as:

$$\sum_{k\in\mathcal{K}}\sum_{e_{st}(v_s)\in e(d_k)} \gamma_{ij}^{st,k}\varepsilon_{v_s}d_k + \gamma_{ij}^{st,k}\varepsilon_{v_s}\sigma_k\hat{d}_k \le b(\mathcal{L}_{ij}) \tag{4.13}$$

where, for any $\sigma_k (k\in J)$, the maximization deviation part is shown as:

$$\max_{\sigma \in U, J \in \mathcal{K}, |J| \leq \Gamma} \left\{ \sum_{k \in J} \sum_{e_{st}(v_s) \in e(d_k)} \gamma_{ij}^{st,k} \varepsilon_{v_s} \sigma_k \hat{d}_k \right\} \tag{4.14}$$

The corresponding robust counterpart of (4.13) is shown as:

$$\begin{cases} \sum_{k \in \mathcal{K}} \sum_{e_{st}(v_s) \in e(d_k)} \gamma_{ij}^{st,k} \varepsilon_{v_s} \bar{d}_{st}^k + \Psi \sum_{k \in \mathcal{K}} \sum_{e_{st} \in e(d_k)} \varsigma_{ij}^{st,k} + \Gamma z_{ij} \leq b(\mathcal{L}_{ij}) \\ \varsigma_{ij}^{st,k} + z_{ij} \geq \hat{d}_k \gamma_{ij}^{st,k} \\ \varsigma_{ij}^{st,k} \geq 0, z_{ij} \geq 0 \quad \forall \mathcal{L}_{ij} \in L, \forall k \in \mathcal{K} \end{cases} \tag{4.15}$$

Notice that the robust formulation can be transformed into the nominal case when $\Psi = 0$ and $\Gamma = 0$. Consider the deterministic formulation (4.6.1) - (4.6.9) with the nominal traffic demands as the nominal case.

Moreover, the objective function of the robust formulation is represented as:

$$max \quad g_r = \omega_u f_u + \omega_e f_e + \omega_m f_m, \tag{4.16}$$

for the robust formulation, constraints (4.12) and (4.15) are the equivalence of the constraints (4.6.2)-(4.6.4), which leverage the Γ and Ψ perturbations of uncertain variables for traffic demands.

Thus, the integrated robust formulation is consisted of (4.16), (4.6.5) - (4.6.9), and the counterparts (4.12) and (4.15) substituting for the constraints (4.6.2) - (4.6.4).

4.4 Algorithms of Service-aware Design Policy

Theoretically, the optimal design solutions can be obtained by calculating the optimization models presented above. However, the proposed NSDP is an Integer Linear Programming (ILP) VNE-type problem, which is NP-hard. Its computational time may be extremely long when its input size is large [9]. Thus, to balance the computational effort and the quality of solutions, an effective heuristic algorithm is proposed, the Service-aware Network Slicing Design Policy (S-NSDP) algorithm, which is inspired by the MOPSO algorithm [55].

The S-NSDP algorithm we proposed is aimed at 1) utilizing system resources efficiently with fluctuated traffic demands, 2) satisfying the various service requirements of different slice requests, 3) guaranteeing the concordant coexistence of different kinds of slices belonging to various use cases in a certain underlying infrastructure.

4.4.1 Basic Concepts of Algorithms

Assume that $\mathcal{S}$ indicates the set of slices, which consists of $\mathcal{K}$ slice service instances. Each $q_k(q_k \in \mathcal{S}, k \in \mathcal{K})$ may have more than one possible design solutions. The design solutions are composed of two parts: the design records $\mathcal{A}_v(q_k)$ of VNFs and the design records $\mathcal{A}_e(q_k)$ of virtual links. In particular, $\mathcal{A}_e(q_k)$ only can be obtained after $\mathcal{A}_v(q_k)$ is determined.

The set $\mathcal{A}(q_k)$ of candidate solutions of q_k consists of $\mathcal{A}_v(q_k)$ and $\mathcal{A}_e(q_k)$. To be specific, $a*b$ computing resource constraints are formulated when q_k consists of a VNFs and the physical infrastructure contains b physical nodes. Besides, $m*n$ bandwidth resource constraints are figured when there are n physical paths between the corresponding physical nodes in $\mathcal{A}_v(q_k)$, to deploy q_k with m virtual links.

In this work, a $\mathcal{K}$-dimensional search space is assumed to obtain the final design solution of $\mathcal{S}$. Specifically, a particle swarm X is defined, which is composed of $\mathcal{M}$ particles. Each particle $X_\eta(\eta \in \mathcal{M})$ of X is expressed as:

$$X_\eta = [x_{\eta,1}, x_{\eta,2}, ..., x_{\eta,k} ..., x_{\eta,\mathcal{K}}] \tag{4.17}$$

where $x_{\eta,k}$ is the k^{th} element of X_η, the value of $x_{\eta,k}$ is the index into the sequence of the set $\mathcal{A}(q_k)$ containing candidate design solutions of q_k. X_η is considered as an integrated design solution of $\mathcal{S}$.

The initial positions of X have a decisive influence on the final design solutions. The X should be first initialized in its search space. The details of the initialization algorithm of particle swarms are presented in Algorithm 4. To be specific, the set

Algorithm 4 Algorithm of Particle Swarm Initialization

```
Input:
    $\mathcal{S}$: the set of slice service instances;
Output:
    $X$: the set of initial particle swarm;
 1: for $q_k(\forall k \in \mathcal{K})$ do
 2:     get $\mathcal{A}(q_k) = \{\mathcal{A}_v(q_k), \mathcal{A}_e(q_k)\}$ based on the G-SP algorithm;
 3:     for $y \in \mathcal{A}(q_k)$ do
 4:         for $l_{ij} \in y$ do
 5:             $\beta_y(q_k) = +\delta(l_{ij})$;
 6:         end for
 7:         rank $\mathcal{A}(q_k)$ based on $\beta_y(q_k)$ as $\mathcal{L}(q_k)$;
 8:     end for
 9: end for
10: if $|\mathcal{L}(q_k)| < \mathcal{M} \quad \& \quad \mathcal{L}(q_k) \neq \varnothing$ then
11:     expand $\mathcal{L}(q_k)$ with size $\mathcal{M}$ by collecting $\mathcal{M}$ elements in $\mathcal{A}(q_k)$ cyclically in
        order;
12: end if
13: for $x_{\eta,k}(\forall \eta \in \mathcal{M}, \forall k \in \mathcal{K})$ do
14:     if $|\mathcal{L}(q_k)| \geq \mathcal{M}$ then
15:         initialize $x_{\eta,k}$ as the index $\eta$ into the sequence of $\mathcal{L}(q_k)$;
16:     else
17:         initialize $x_{\eta,k}$ as the random index into the sequence of $\mathcal{L}(q_k)$;
18:     end if
19:     add $x_{\eta,k}$ into $X_\eta$;
20: end for
21: return $X$
```

$\mathcal{A}(q_k)$ of candidate solutions is updated as $\mathcal{L}(q_k)$ by ranking the total delay $\beta_y(q_k)$ of its included solution $y(y \in \mathcal{A}(q_k))$, which is shown as Lines $1-12$. When the size of $\mathcal{L}(q_k)$ is larger than or equal to $\mathcal{M}$, the value of each element $x_{\eta,k}$ of X_η is initialized as the index into the sequence of the η^{th} solution in $\mathcal{L}(q_k)$; otherwise, the initialization of $x_{\eta,k}$ is the index into the sequence of a random solution. The initialization processes of particles are represented as Lines $13-21$.

After the initialization of the X, feasible design solutions can be calculated by $\mathcal{N}$ times iterations. $\mathcal{B}_{\eta,k}$ denotes the personal optimal solution of $x_{\eta,k}$ obtained so far, and $\mathcal{B}_{g,k}$ represents the global optimal solution of the whole particle swarm X. Particles will fly towards $\mathcal{B}_{g,k}$ while searching for their personal optimal solutions during each iteration. The velocity of the particle X_η will update towards $\mathcal{B}_\eta$ and $\mathcal{B}_g$, respectively, based on its current position, personal optimal position $\mathcal{B}_\eta$ and global optimal position $\mathcal{B}_g$. That is, $\mathcal{V}_{X_\eta \to \mathcal{B}_\eta}$ and $\mathcal{V}_{X_\eta \to \mathcal{B}_g}$. The new velocity of X_η can be updated as:

$$(\mathcal{V}_\eta(t), \mathcal{V}_{X_\eta \to \mathcal{B}_\eta}, \mathcal{V}_{X_\eta \to \mathcal{B}_g}) \Rightarrow \mathcal{V}_\eta(t+1) \tag{4.18}$$

where $\mathcal{B}_\eta$ is represented by:

$$\mathcal{B}_\eta = [\mathcal{B}_{\eta,1}, \mathcal{B}_{\eta,2}, ..., \mathcal{B}_{\eta,k}, ..., \mathcal{B}_{\eta,\mathcal{K}}], \tag{4.19}$$

and $\mathcal{B}_g$ is denoted by:

$$\mathcal{B}_g = [\mathcal{B}_{g,1}, \mathcal{B}_{g,2}, ..., \mathcal{B}_{g,k}, ..., \mathcal{B}_{g,\mathcal{K}}]. \tag{4.20}$$

Let $\mathcal{V}_\eta = (\mathcal{V}_{\eta,1}, \mathcal{V}_{\eta,2}, ..., \mathcal{V}_{\eta,k}, ..., \mathcal{V}_{\eta,\mathcal{K}})$ denote a velocity set of each element $x_{\eta,k}$ of X_η. Each $\mathcal{V}_{\eta,k}$ is defined as:

$$\begin{aligned}\mathcal{V}_{\eta,k}(n+1) =& \omega \cdot \mathcal{V}_{\eta,k}(n) + r_1 \cdot w_l C_1 \cdot (\mathcal{B}_{\eta,k}(n) - x_{\eta,k}(n)) + \\ & r_2 \cdot w_g C_2 \cdot (\mathcal{B}_{g,k}(n) - x_{\eta,k}(n))\end{aligned} \tag{4.21}$$

where $\eta \in \mathcal{M}$, $k \in \mathcal{K}$, $n \in \mathcal{N}$, and ω denotes the inertia weight. r_1 and r_2 represent two random numbers which are uniformly distributed in [0,1]. w_l and w_g denote local parameter and global parameter, respectively. C_1 and C_2 indicate the learning factors to adjust the convergence of the algorithm, they are defined by the size of $\mathcal{A}(q_k)$.

Regarding the new position $X_\eta(n+1)$ of the η^{th} particle, it can be calculated by the velocity $\mathcal{V}_\eta(n+1)$ and its current position $X_\eta(n)$. The regeneration of each element $x_{\eta,k}$ of X_η is shown as:

$$x_{\eta,k}(n+1) = x_{\eta,k}(n) + \mathcal{V}_{\eta,k}(n+1). \tag{4.22}$$

The details that the particle position of X_η updates during each iteration are shown in Algorithm 5.

Algorithm 5 Algorithm of Particle Position Update

```
Input:
    $X_\eta(n)$: the current position of $X_\eta$,
    $\mathcal{B}_\eta(n)$: the personal optimal position,
    $\mathcal{B}_g(n)$: the global optimal position;
Output:
    $X_\eta(n+1)$: the new position of $X_\eta$;
1: for $x_{\eta,k} \in X_\eta(n), \forall k \in \mathcal{K}$ do
2:     update $\mathcal{B}_{\eta,k}(n)$ and $\mathcal{B}_{g,k}(n)$;
3:     $\mathcal{V}_{\eta,k}(n), \mathcal{B}_{\eta,k}(n), \mathcal{B}_{g,k}(n) \Rightarrow \mathcal{V}_{\eta,k}(n+1)$;
4:     $x_{\eta,k}(n), \mathcal{V}_{\eta,k}(n+1) \Rightarrow x_{\eta,k}(n+1)$;
5:     if $x_{\eta,k}(n+1) < 0$ then
6:         $x_{\eta,k}(n+1) = 0$;
7:     else if $x_{\eta,k}(n+1) \geq M$ then
8:         $x_{\eta,k}(n+1) = \text{random.randint}(1, |\mathcal{A}(q_k)|)$;
9:     else
10:        $x_{\eta,k}(n+1) = \lceil x_{\eta,k}(n+1) \rceil$;
11:    end if
12: end for
13: return $X_\eta(n+1)$
```

4.4.2 Detailed Description of Algorithms

In the S-NSDP algorithm, we assume that traffic demands will vanish when their source and destination are placed in the same infrastructure node. And q_k will be removed from the infrastructure when its TTL o_k expires. Besides, the traffic demand d_k routing q_k may fluctuate after it passes the set of predefined VNFs v, which is shown as Fig. 4.1.

The solution of the versatile design policy should guarantee the various service requirements in different use cases. It can be obtained by updating the set of candidate design solutions. In particular, the candidate solutions are evaluated by their fitness values. And the fitness value of each candidate solution is calculated by the general design objective function. Thus, the fitness function F of each particle $X_\eta (\forall \eta \in \mathcal{M})$ is defined to evaluate the quality of each candidate design solution, which can be expressed as:

$$F(X_\eta) = \omega_e R_e + \omega_m R_m + \omega_u R_u \tag{4.23}$$

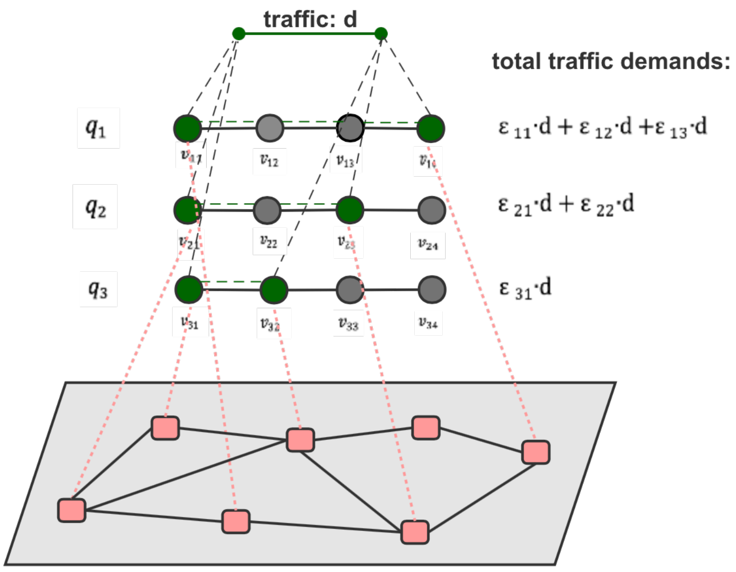

traffic: d
total traffic demands:
q_1
q_2
q_3
v_{11} v_{12} v_{13} v_{14}
v_{21} v_{22} v_{23} v_{24}
v_{31} v_{32} v_{33} v_{34}
$\varepsilon_{11}\cdot d + \varepsilon_{12}\cdot d + \varepsilon_{13}\cdot d$
$\varepsilon_{21}\cdot d + \varepsilon_{22}\cdot d$
$\varepsilon_{31}\cdot d$

Fig. 4.1 Traffic demands in different slice requests through the same underlying infrastructure.

where ω_e, ω_m and ω_u are the weighting coefficients of eMBB, mMTC and uRLLC slices, respectively. Besides, R_e denotes the average utilization of bandwidth resources, which is shown as:

$$R_e = \frac{\sum_{k\in\mathcal{K}}\sum_{e_{st}(v_s)}\sum_{\mathcal{L}_{ij}} \varepsilon_{v_s} d_k}{\sum_{\mathcal{L}_{ij}} b(\mathcal{L}_{ij})}; \tag{4.24}$$

R_m represents the average utilization of CPU and memory resources, and it is defined by:

$$R_m = \frac{\sum_{p_i\in P}\sum_{v_s\in v} x_{v_s}(c_{v_s}+m_{v_s})}{\sum_{p_i\in P}(c_{p_i}+m_{p_i})} + \frac{\sum_{p_i\in P}\sum_{v_s\in v}\sum_{k\in\mathcal{K}}\{(c_{\lambda_{c_s}}+m_{\lambda_{c_s}})\varepsilon_{v_s}d_k\}}{\sum_{p_i\in P}(c_{p_i}+m_{p_i})}; \tag{4.25}$$

R_u indicates the normalization value of the latency of all physical paths when the design solution X_η is implemented, its definition is shown as:

$$R_u = 1 - \sum_{\mathcal{L}_{ij}}\{\frac{\sum_{k\in\mathcal{K}}\sum_{e_{st}(v_s)\in e}\varepsilon_{v_s}d_k\xi_{ij}}{\mathcal{C}} + \frac{\kappa_{ij}+\sum_{k\in\mathcal{K}}\sum_{p_i\in P}(\iota_{p_i}+\sum_{v_s\in v}\iota_{v_s})}{\mathcal{C}}\}. \tag{4.26}$$

The details of the S-NSDP algorithm are presented in Algorithm 6. The number of iterations $\mathcal{N}$ and the size $\mathcal{M}$ of the particle swarm X need to be initialized firstly. Each particle $x_{\eta,k}$ in X is initialized by Algorithm 4. An initial velocity set $\mathcal{V}_\eta$ is created and the value of $\mathcal{V}_{\eta,k}$ is initialized as 1. $\mathcal{B}_\eta$ and $\mathcal{B}_g$ are initialized, where their initial fitness values F_η and F_g of the particle X_η are set to 0, respectively. For each iteration $n(n \in N)$, the particle $x_{\eta,k}$ is updated by Algorithm 5, and $x_{\eta,k} = 0$ indicates that the implementation of q_k is not successful. Next, evaluate the quality of X_η by its fitness value, and update F_η and F_g as the optimal value in the historical records. Afterwards, update $\mathcal{B}_\eta$, F_η, $\mathcal{B}_g$ and F_g until all iterations are completed. The final design solution $\mathcal{D}$ is considered as $\mathcal{B}_g$. Besides, traffic demands in $\mathcal{S}$ are fluctuated and their TTLs o_k may be different. The total number of time slots is defined as $\mathcal{O}$. When each slot ends, the occupied records of $\mathcal{S}$ are traversed. q_k will be removed from the infrastructure when its TTL expires, meanwhile, the occupied resources of the slice will also be released. It may cause the occupancy status of the substrate network is varying.

In the S-NSDP algorithm, $O(|\mathcal{A}(q_k)| \cdot (1+\log_2|\mathcal{A}(q_k)|) + \mathcal{M} \cdot (1+\mathcal{N}) + \mathcal{O})$ time is required to obtain the design solution of each slice request. The occupied records of the infrastructure will be updated once each time slot ends.

4.5 Performance Evaluation

In this section, the details of simulation scenarios are first described. Then, the evaluation results are analyzed to validate the proposed formulations and algorithms of the service-aware NSDP.

In this chapter, the deterministic formulation is defined as the so-called nominal case, which considers the nominal values of traffic demands. The robustness formulation is investigated by comparing it with the nominal case. Specifically, the impact on the objective value of the nominal case is analyzed in terms of the objective gaps. The definition of the gap percentage can be defined by:

$$gap = \frac{|g_d - g_r|}{g_d} \cdot 100\%. \tag{4.27}$$

Algorithm 6 Service-aware Network Slicing Design Policy Algorithm

```
Input:
    $G$: the physical network;
    $\mathcal{S}$: the set of slice service instances;
Output:
    $\mathcal{D}$: the set of design solutions of $\mathcal{S}$;
    $F_g(\mathcal{D})$: the global optimal fitness value;
1: initialize $\mathcal{N}$, $\mathcal{M}$;
2: initialize the population of the particle swarm $X$ by Algorithm 4;
3: initialize the velocity set $\mathcal{V}_\eta(\eta\forall\mathcal{M})$ of each particle in $X$ as $\mathcal{V}_\eta = ones[\mathcal{K}]$;
4: evaluate the initial $X$;
5: initialize $\mathcal{B}_\eta$, $F_\eta$, $\mathcal{B}_g$, $F_g$;
6: while $n \in \mathcal{N}$ do
7:     for $X_\eta(\eta\forall\mathcal{M})$ do
8:         for $x_{\eta,k}(k\forall\mathcal{K})$ do
9:             if $x_{\eta,k} \notin \mathcal{A}_{q_k}$ then
10:                $x_{\eta,k} = 0$;
11:            end if
12:            update $x_{\eta,k}$ based on Algorithm 5;
13:        end for
14:        evaluate $F(X_\eta)$;
15:        update $\mathcal{B}_\eta$, $F_\eta$;
16:        update $\mathcal{B}_g$, $F_g$;
17:    end for
18:    increase the iteration counter $n$;
19: end while
20: $\mathcal{D} = \mathcal{B}_g$;
21: for $\mathcal{O}$ do
22:    traverse the occupied records of VNFs and virtual links of $\mathcal{S}$ based on $\mathcal{D}$;
23:    if $o_k = 0$ then
24:        remove $q_k$ and release the resource it occupied;
25:    else
26:        $o_k = o_k - 1$;
27:    end if
28:    update $F_g(\mathcal{D})$;
29: end for
30: return $\mathcal{D}$, $F_g(\mathcal{D})$
```

The present heuristic algorithms are evaluated from different aspects, such as the average CPU and memory resource utilization, average bandwidth resource utilization and acceptance ratio, compared with the following two algorithms: the Random Greedy algorithm and the PARAA (Profit-Aware Resource Allocation) algorithm.

The approach of randomly drawing samples from the candidate set $\mathcal{A}(q_k)$ is defined as a Random Greedy method. $\mathcal{A}(q_k)$ consists of the candidate design solutions for each $q_k(\forall k \in K)$. Specifically, the design solutions of the VNFs of each slice request are sampled from $\mathcal{A}(q_k)$. And virtual links are placed on the physical paths with the most sufficient resources. The deployments of subsequent slice requests may fail when a certain system resource is exhausted. It can be considered as a greedy comparison algorithm to evaluate our proposed algorithms.

In addition, the PARAA algorithm [110] aims to solve the resource allocation problem by using the BPSO (Binary Particle Swarm Optimization) methodology [111] based on the costs and quality of service requirements of different network slice cases for the 5G sliced networks. As this algorithm focuses on the radio access network, it is developed here by considering the bandwidth resources and costs, making it comparable with the present algorithm.

4.5.1 Simulation Setup

The deterministic and robust models are both implemented using MATLAB with the programming solver Cplex 12.9.0 integrated. The robust model is carried out by ROME [112] that is a MATLAB toolkit for modeling and solving robust optimization problems. Heuristic algorithms are achieved using Python 3.7. The simulations are performed on a Linux laptop with Eight CPU cores of 1.9GHz and 16GB RAM.

Regarding the infrastructure network, a sample network topology POLSKA of SNDlib [113] is considered, which consists of 12 nodes and 18 links. The values of Δ of all physical nodes are set to 1. The CPU and memory capacities of physical nodes are respectively sampled from the tuples $(80, 110, 150, 180)$ and $(100, 140, 160, 200)$ at random, with a probability of $(0.2, 0.3, 0.3, 0.2)$. The bandwidth capacity of physical links has the uniform distribution of $U[10, 30]$. The number of VNFs in each slice request is uniformly drawn from the tuple $(2,3,4,5,6)$ with a probability of 0.2. For each set of traffic demands $D_k(\forall k \in K)$, its source $s(D_k)$ and destination $h(D_k)$ are collected from P while ensuring the length of each virtual link is at least 1. All physical nodes have the same priority to host different types of VNFs. Further, it is assumed that each slice service instance of specific use case where the set of VNFs may be considered as the subset of $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$ that consists of 8 types of VNFs. For instance, the set of VNFs of q_1 is given as $v = \{v_1, v_3, v_5, v_7\}$ in which the set of end-to-end traffic demands traversing VNFs is represented as: $\{(s, v_1), (v_1, v_3), (v_3, v_5), (v_5, v_7), (v_7, h)\}$.

The sizes of CPU and memory resource blocks c_{v_s} and m_{v_s} of each VNF $v_s(\forall v_s \in V)$ both have the uniform distribution of $U[2,7]$. The unit resource coefficients $c_{\lambda_{v_s}}$ and $m_{\lambda_{v_s}}$ of each VNF v_s take value from the uniform distribution of $U[1,3]$. The VNF processing delay ι_{v_s} of $v_s \in V$ and the physical node processing delay ι_{p_i} of $p_i \in P$ are both set to 1. And the maximum path transmission delay ξ_{ij} of $\mathcal{L}_{ij}$ equals to 3. Assume that the elastic coefficient ε_{v_s} of each VNF v_s is sampled from the uniform distribution of $U[0.5,2]$, which means the traffic demands can fluctuate from halving to doubling after routing through them. The maximum TTL of each slice is set to 8. And the constant $\mathcal{C}$ equals to 18.

Slice service instances are generated with an increasing number of requests $\mathcal{K} = 6, 12, 24, 36, 48, 60$, every instance of a given topology contains a set of traffic demands. 10 data sets are generated for each type of slice requests. The values of each data set are sampled from the uniform distributions $U[1.5,4]$, $U[1,2.5]$ and $U[0.1,0.5]$, respectively, for the traffic demands of eMBB, mMTC and uRLLC slices. The nominal values of traffic demands $(\bar{d_k}, \forall k \in \mathcal{K})$ are calculated as the arithmetic mean over the generated data sets, and the positive perturbation $\hat{d_k}$ is determined by the standard deviation. The length of virtual links for eMBB and mMTC slices is respectively set from the tuple $(3,4,5,6)$ uniformly at random with an equal probability, similarly, and the length of uRLLC slices is sampled from the tuple $(1,2,3)$.

The number of eMBB, mMTC and uRLLC slices are defined by $\omega_e\mathcal{K}$, $\omega_w\mathcal{K}$ and $\omega_u\mathcal{K}$, respectively, and $\omega_e, \omega_w, \omega_u \in [0,1]$. We set different application scenarios to illustrate the results of the S-NSDP algorithm with different values of ω_e, ω_w and ω_m, which are $\omega_u : \omega_m : \omega_e = \{1/3 : 1/3 : 1/3\}$, $\{1/2 : 1/6 : 1/3\}$, $\{1/3 : 1/6 : 1/2\}$, $\{1/6 : 1/2 : 1/3\}$. Besides, $\omega_u : \omega_m : \omega_e = \{1 : 0 : 0\}, \{0 : 1 : 0\}, \{0 : 0 : 1\}$ respectively indicates the application scenario of uRLLC, mMTC and eMBB slices. The inertia weight ω of the S-NSDP algorithm is set to 0.729 [101]. w_l and w_g are both assumed as 0.5. In addition, we adopt $\Gamma = 1, 2, 3$ and $\Psi = 0.25, 0.5, 0.75, 1, 1.25, 1.5$.

For the parameter setting of the PARAA algorithm, the InP bandwidth capacity is also sampled from $U[10, 30]$, each network operator (NO) requires a specific type of slices with the number of NO as 3. With each network slice attached to a user, the number of slices of each NO is set to $2, 4, 8, 12, 16, 20$, respectively. The cost of each slice request is assumed to sample from a uniform distribution of $U[1, 10]$. The controlling parameters W_{min}, W_{max}, c_1 and c_2 are set to 0.2, 0.9, 0.2, 0.2, respectively. For both our proposed algorithm and the PARAA algorithm, the number of particles M is set to 50, and the threshold of iteration N is given as 300.

4.5.2 Evaluations Results

Fig. 4.2 shows the impact on the objective value of the deterministic case with nominal traffic demands. It is illustrated by the objective gaps compared with the robustness model considering the uncertain traffic demands. We focus on the scenario where the number of slice requests is $\mathcal{K} = 6$ and $\omega_u : \omega_m : \omega_e = \{1/3 : 1/3 : 1/3\}$. Since the objective gap percentage is achieved almost 100%, larger values of Γ and Ψ are not further reported. For the fixed value of Γ, the objective gap becomes larger with the lager values of Ψ. For instance, $\Gamma = 1$, it can be observed that the gap percentage is 59.67% for $\Psi = 0.5$, and 76.52% for $\Psi = 1$. Besides, the robustness model is related to the value of Ψ. The uncertainty may increase with an increasing Ψ, which leads to a larger gap. When the value of Ψ is less than or equal to 1, the result of the robustness model is more obviously affected by Ψ. For instance, $\Psi = 0.25, 0.5, 0.75, 1$, the gaps remain the same as the value of Γ increases. On the other hand, when the value of Ψ is larger than 1, the result of the robustness model is mainly determined by Γ. For example, when Γ is assumed as 2 or 3, the gaps become larger as 84.9%, 93.39%,

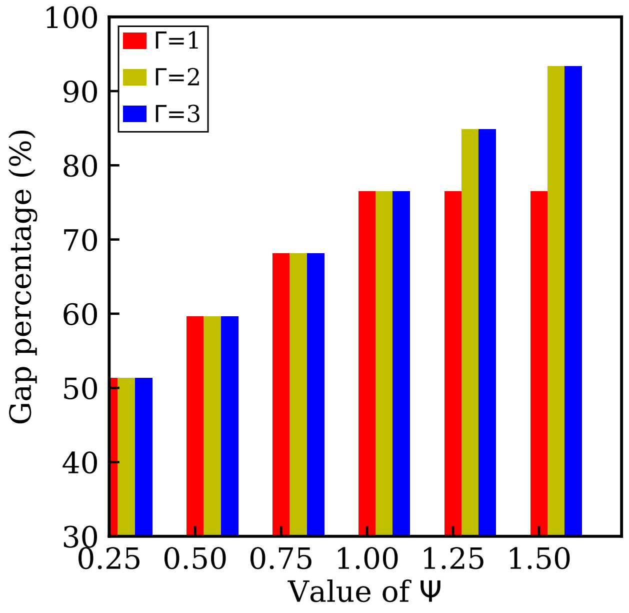


Fig. 4.2 Gap percentages for different values of Ψ under $\Gamma = 1,2,3$.

respectively, for $\Psi = 1.25$ and $\Psi = 1.5$. Thus, the illustrated results indicate that the impact on the objective value of the nominal case is determined by different values of Γ and Ψ. That is, the objective gaps become larger with the increment of the number of deviated traffic demands and their fluctuations. And the gap percentage can achieve almost 100% when Γ is given as 2 or 3.

Fig. 4.3 illustrates the impact on the objective value of the nominal case compared with different application scenarios of the robustness model, where $\omega_u : \omega_m : \omega_e = \{1/3 : 1/3 : 1/3\}, \{1 : 0 : 0\}, \{0 : 1 : 0\}, \{0 : 0 : 1\}$ with $\mathcal{K} = 6$ and $\Gamma = 2$. The gap percentage of mMTC slices is larger than the other two cases because their traffic demands are smallest, which leads to their objective values being susceptible to traffic fluctuations. The values of traffic demands in eMBB slices are greater than those in uRLLC and mMTC slices, which causes the single objective value of eMBB slices regarding bandwidth consumption to be less affected by the same level of fluctuations. For example, for $\Psi = 0.25$, the gap percentage of eMBB slices equals 13.8%, however, the gap percentages of mMTC and uRLLC slices are respectively 47.37%, 39%. Besides, the objective gap percentage of eMBB slices increases linearly since there is no influence of latency and computing resource consumption. We can see that the

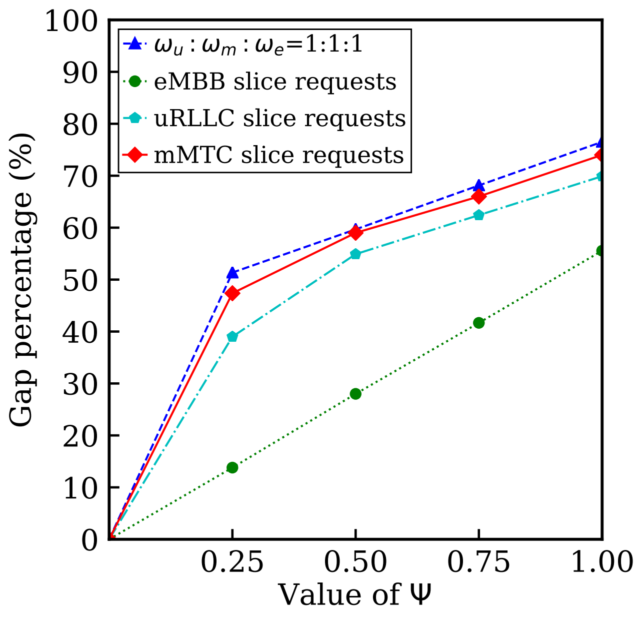


Fig. 4.3 Gap percentages for different values of Ψ ($\Gamma = 2$) of different types of slice requests.

change tendency of the gap percentage of bandwidth consumption for eMBB slices is nearly proportional to the changing trend of the value of Ψ. The values of the gap percentage of eMBB slices are almost multiplied when the values of Ψ increase multiply. For instance, the gap percentage increases from 13.8% to 27.78% when the value of Ψ changes from 0.25 to 0.5; the gap percentage increases from 13.8% to 41.67% when the value of Ψ grows from 0.25 to 0.75.

In Fig. 4.4, we compare the bandwidth utilization of eMBB, mMTC and uRLLC slices over different time units to illustrate their changes. In the present work, the design process of all network slice requests is offline, network slices will be removed from the network when their TTLs expire and no more new slices need to be deployed. The eMBB, mMTC and uRLLC slices are deployed based on their specific design objectives. Each curve indicates the bandwidth utilization of network slice requests in different use cases, and the number of slices for different use cases is set to $\mathcal{K} = 30$, respectively. It can be observed that the ratios of bandwidth utilization gradually decrease along with the increment of the time unit. Because more and more slices are removed from the infrastructure when their TTLs expire, the occupied bandwidth

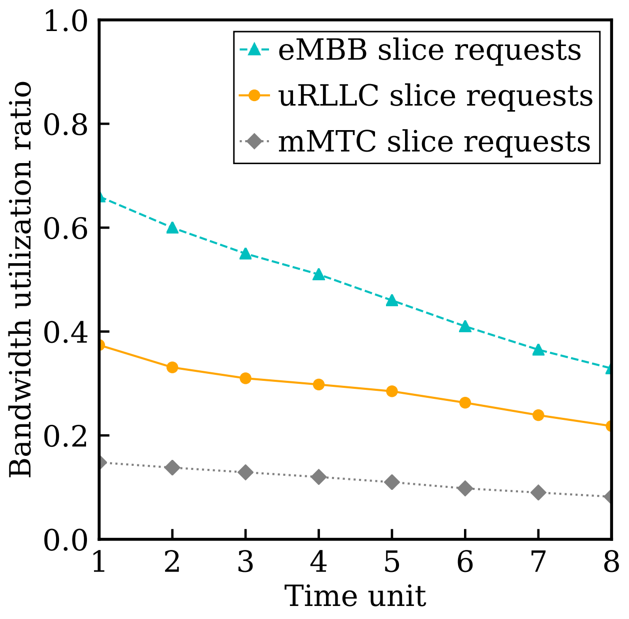


Fig. 4.4 Bandwidth utilization ratios for different time units of different types of slice requests.

resources are released and the bandwidth utilization ratio is decreased. The results also show that the ratios of bandwidth utilization of eMBB slices are higher than the other two use cases, which is caused by their highest values of traffic demands, even though their elastic coefficients take values from the same uniform distribution.

To evaluate the resource utilization of eMBB, mMTC and uRLLC slices respectively, the utilization ratios of CPU and memory resources for the first time slot are shown in Fig. 4.5. Notice that the utilization ratios of CPU and memory resources both gradually increase with the increment of the number of slice requests. Because more resources are required as more VNFs are placed on the infrastructure. Besides, VNFs consume more resources to process traffic demands as they route through more. The resource utilization ratios of CPU and memory resources in uRLLC slices both are the lowest. It is because their specific design objective mainly focuses on the minimum latency so that traffic commodities can remain shortest. That is, less VNFs are occupied, which also leads to less resources are required to process traffic demands. Moreover, the resources consumed by mMTC slices are less than that of eMBB slices since the traffic demands in mMTC slices are much smaller than eMBB slices. For

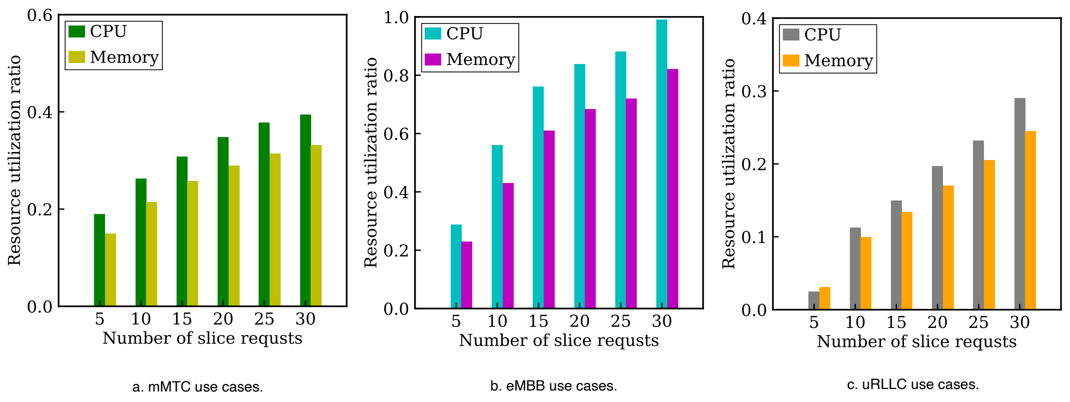


Fig. 4.5 Resource utilization ratios for different number of slice requests: a. mMTC, b. eMBB, c. uRLLC.

instance, for $\mathcal{K} = 15$, the utilization ratios of CPU resources of mMTC slices and eMBB slices are 30.8% and 76%, respectively.

In Fig. 4.6, the acceptance ratios of the first time slot are plotted, regarding the different numbers of slice requests. The acceptance ratios are calculated by the ratio of the number of embedded slices to the whole number of slice requests. To assess the proposed algorithm, four scenario cases with different ω_u, ω_e, and ω_m are implemented. It is observed that the acceptance ratios of the two comparison algorithms and four proposed cases all decrease gradually as the number of slice requests increases. Besides, the acceptance ratios of the proposed cases are higher than that of the two comparison algorithms. The main reason is that the proposed multiple design objectives can coordinate resources consumption from different perspectives, which may lead to fewer failures due to the exhaustion of one type of resources in the infrastructure. It also can be observed that the acceptance ratios of the PARAA algorithm are better than that of the Random Greedy algorithm. Because the PARAA algorithm can not only allocate bandwidth resources optimally but also maximize the system profit. However, the Random Greedy algorithm adopts a greedy scheme to deploy slices so that it may cause more deployment failures due to the exhaustion of infrastructure resources.

In addition, Fig. 4.7 illustrates the average ratios of CPU and memory resource utilization of the six cases. Since the acceptance ratios of the proposed cases are higher compared with the PARAA algorithm and the Random Greedy algorithms, it can be

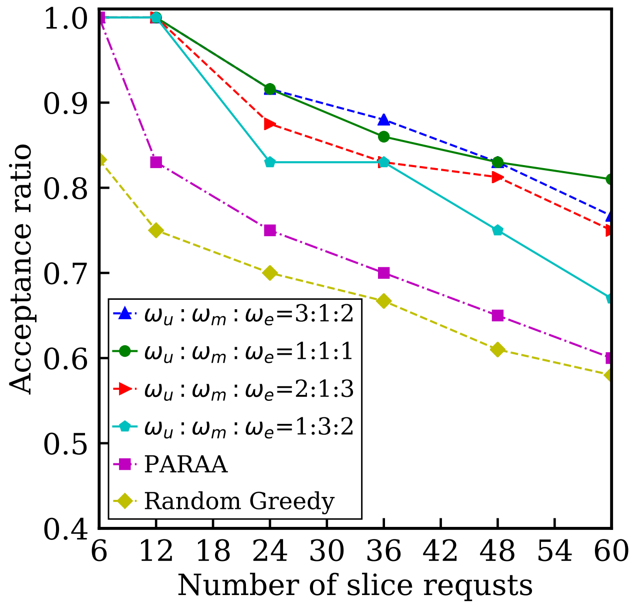


Fig. 4.6 Acceptance ratios for different number of slice requests: a comparison between the proposed algorithm and existing algorithm.

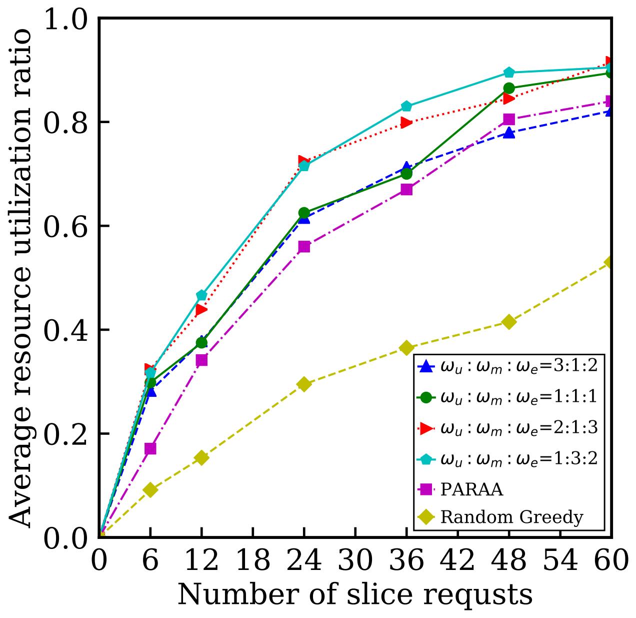


Fig. 4.7 Average ratios of resource utilization for different number of slice requests: a comparison between the proposed algorithm and existing algorithm.

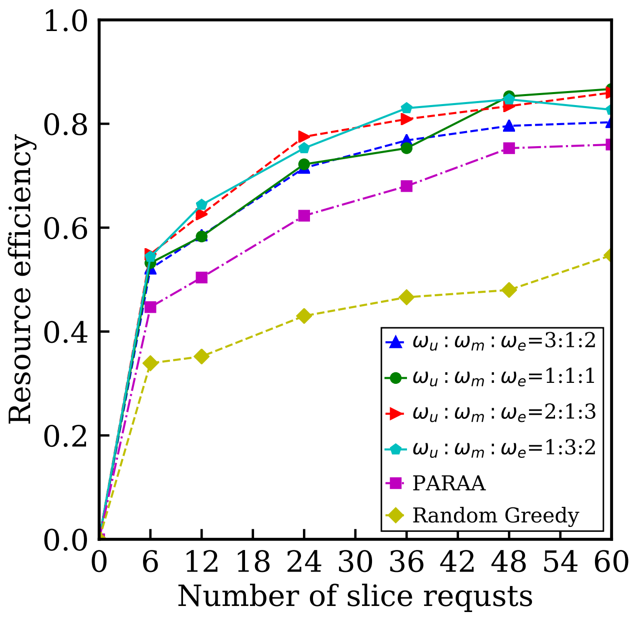


Fig. 4.8 Resource efficiency for different number of slice requests: a comparison between the proposed algorithm and existing algorithm.

seen that their resource utilization ratios are also higher than that of the comparison algorithms. In Fig. 4.8, the resource efficiencies of different cases are compared over the first time slot. The resource efficiency is determined by the mean of the acceptance ratio and the ratio of resource utilization. That is to say, it is not only related to the number of embedded slices but also is affected by the resource consumption. It can be seen that the resource efficiencies of the proposed cases are better compared with the other two algorithms. Furthermore, the resource efficiency gradually increases when the number of slices requests becomes larger. Specifically, all curves rise sharply when the number of slice requests changes from 0 to 6. And more than 60% infrastructure nodes are occupied when $\mathcal{K} = 6$, which can be observed in Fig. 4.9. All infrastructure nodes are occupied when $\mathcal{K}$ is larger than 24.

Regarding the case of $\omega_m : \omega_u : \omega_e = \{1/3 : 1/3 : 1/3\}$, the utilization ratios of bandwidth resources are plotted respectively for different time slots in Fig. 4.10. In this figure, each line indicates the bandwidth utilization of different numbers of slice requests. It can be observed that more bandwidth resources are consumed as the number of slice requests increases. Particularly, 40% to 60% bandwidth resources

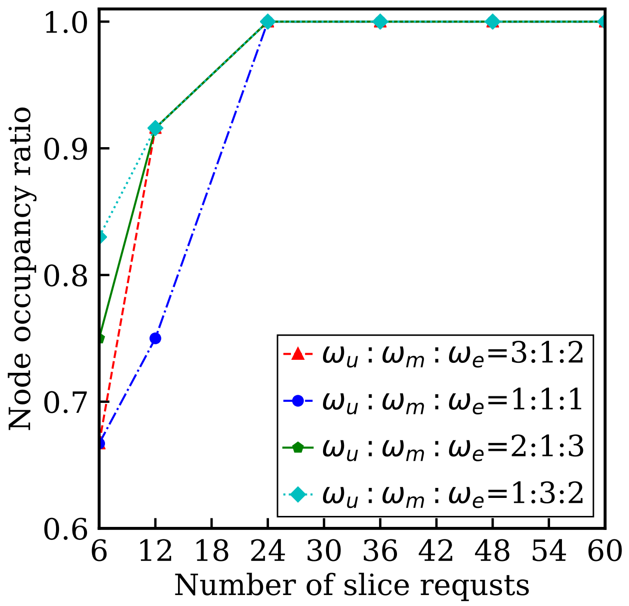


Fig. 4.9 Node occupancy ratios for different number of slice requests: a comparison of the proposed algorithm with different values of ω_e, ω_m, ω_u.

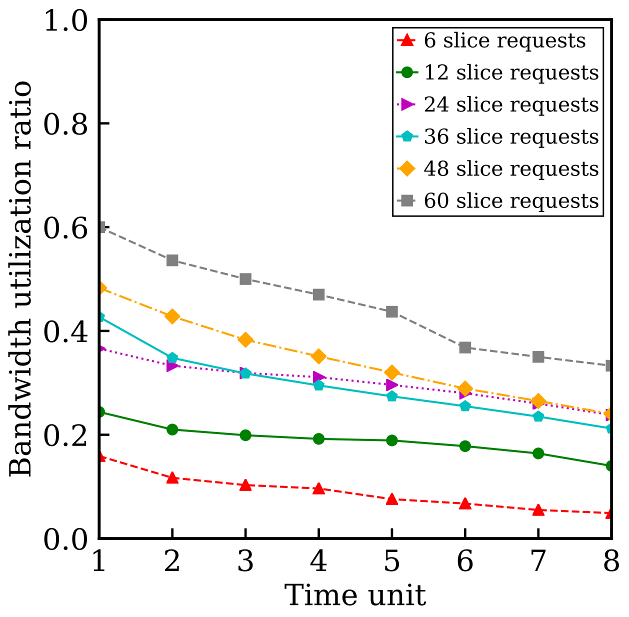


Fig. 4.10 Bandwidth utilization ratios for different time units ($\omega_m : \omega_u : \omega_e = 1:1:1$) of different number of slice requests.

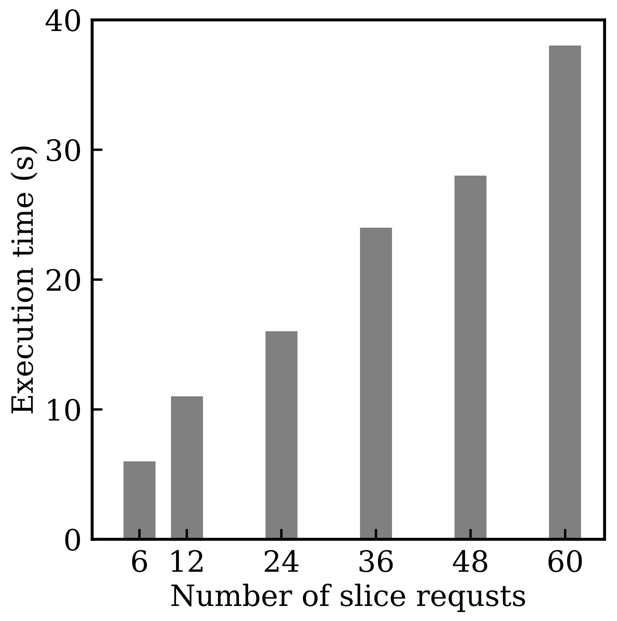


Fig. 4.11 Execution time with the increment of the number of slice requests.

are consumed over the different 8 time slots when $\mathcal{K} = 60$. The utilization ratios of bandwidth resources are decreased with the increment of the time slot since the TTLs of slice requests gradually expire and they will be removed from the infrastructure.

Further, Fig. 4.11 illustrates the execution time of our proposed algorithm for different numbers of slice requests. The average execution time of the four scenarios with different ω_u, ω_e, and ω_m increases as the number of slice requests increases. Specifically, the average execution time of the design process for each network slice request is 0.745s.

4.6 Conclusions

Network slicing facilitates the implementation of 5G networks. It can not only customize virtual networks in the form of isolated slices but also satisfy different resource requirements for various use cases to provide services. In this chapter, the service-aware NSDP problem is investigated by considering fluctuated traffic demands for different use cases, including eMBB, uRLLC and mMTC.

To solve the service-aware NSDP, firstly, multiple design objectives are proposed. Then, the deterministic optimization model for nominal traffic demands is formulated. And it is extended into a robust formulation with uncertain traffic demands to provide robust communication services. Besides, a heuristic algorithm of the service-aware NSDP is presented to balance the computational cost and quality of solutions. It is shown that the two formulations can be solved in small scale networks. And the impact on the objective value of the deterministic model are varied with different robustness coefficients Γ and Ψ. Furthermore, the simulations validate the performance of the S-NSDP algorithm, presenting improved results in terms of resource efficiency compared to the existing work.

In the next chapter, an energy-aware NSDP problem is explored by adopting the Deep Reinforcement Learning method, which can avoid the disadvantage of heuristic algorithms of falling into local optimal.

Chapter 5

Energy-aware Design Policy of End-to-End Network Slicing using Deep Reinforcement Learning

5.1 Introduction

The carbon emission problem has been attracting extensive attention, together with the energy shortage problem in some countries. Regarding the high power of the devices in 5G technology, their energy consumption should be considered in implementations. Network slicing is a core technology to implement the 5G or beyond network systems, and it is significant to achieve its energy-aware deployment. Network slicing can support a wide range of network services with various performance requirements [1]. Specifically, the underlying infrastructure can be divided into several logical, virtual and isolated network slices.

In the 5G network slicing technology, the Network Slicing Design Problem (NSDP) as an emerging research topic has been developed, and it origins from the Virtual Network Embedding (VNE) and Service Function Chaining (SFC) techniques. The NSDP aims to deploy various network slices onto the physical network and allocate their required resources efficiently, such as computational, storage and networking resources, while satisfying various Service Level Agreements (SLAs) of different slice tenants.

The NSDP considering energy cost should be investigated to achieve efficient energy utilization and guarantee a balance between energy consumption and the acceptance ratio of the network slices deployed onto the physical infrastructure. However, the energy cost of the deployment processes of network slices has not been thoroughly considered, although the VNE and SFC problems regarding the energy issues have been studied widely. For instance, Su *et.al* [114] proposed an energy cost model and formulated an energy-aware VNE problem as an integer linear programming problem by considering the energy consumption of physical nodes and links. Lin *et.al* [115] provided an energy-aware SFC embedding scheme in a dynamic traffic scenario, specifically, they introduced three heuristic algorithms for achieving energy savings by switching off idle devices.

Thus, it is necessary to investigate an Energy-Aware Network Slicing Design Problem (EA-NSDP) by incorporating energy issues into the design processes of network slices. Generally, an optimization problem of the NSDP can be solved by several existing techniques, including exact-based approaches and heuristic-based algorithms. Both of them can obtain acceptable results in most cases [116]. However, these methods may be limited to some extent to solve the optimization problem, because there are still disadvantages to these methods. Specifically, they are challenging to deal with the NSDP within a polynomial computation time due to its NP-hard characteristic in the practical context of large-scale networks, although the exact-based approaches may be effective for small-scale networks. Besides, convergence issues are reported for heuristic-based algorithms (e.g. Particle Swarm Optimization algorithm [111], Evolutionary algorithm [117] and Tabu Search algorithm [118]), and they fail to benefit from historical experience.

Alternatively, the Deep Reinforcement Learning (DRL) as a promising approach for solving the problems in network slicing has begun to been studied [119–122]. For example, in Reference [122], the authors proposed two DRL-based algorithms to determine how to orchestrate network slices intelligently from the perspective of slice isolation and cloud-edge collaboration. In general, the DRL methods can integrate an automatic optimization framework to address the decision-making problems effectively with the benefit of historical experience by adopting the Reinforcement Learning (RL) architecture as a learning agent and employing the Deep Learning (DL) technology as the approach to feature extractions.

In this chapter, a DRL framework is developed to solve the EA-NSDP inspiring from the Neural Combinatorial Optimization paradigm [60]. To be specific, the energy efficiency is investigated from two aspects: the energy consumption of physical nodes and the energy consumption of physical links [114]. The EA-NSDP is formulated as a Markov Decision Process (MDP) problem, and the problem can be solved based on DRL. In particular, we exploit the Advantage Actor-Critic (A2C) algorithm architecture [86] to train the policy network for sampling the candidate solutions, and a search strategy is proposed to refine the parameters of the policy network and determine the final design solutions. The main contributions of this chapter are summarized as follows:

- An energy-aware objective function is designed for the NSDP to improve the energy efficiency, including the node energy consumption and link energy consumption. Specifically, the energy consumption E_N of physical nodes consists of the switching power cost of host nodes, the basal power cost and demand power cost of network nodes. Besides, the energy consumption E_L of physical links contains the demand power consumption of network links and the switching power cost of forwarding nodes.

- The proposed EA-NSDP is modeled as a MDP problem with a set of reinforcement learning elements, including a state space, a finite action set and a reward function. In particular, the reward function is defined based on the link energy consumption by network slice requests, which can establish a connection between the design processes of VNFs and virtual links in network slices.

- The A2C algorithm architecture is employed to train the proposed algorithm framework. The parameterized policy network is regarded as an actor network, and its parameters are optimized based on a policy gradient method under the guidance of a critic network. The policy network is implemented by leveraging the pointer network architecture with an attention mechanism, which contains two RNN networks: encoder and decoder. Besides, a search strategy is presented to update and refine the parameters of the actor network and select the final design policy of network slices. The present model architecture can provide flexibility and scalability in terms of the varying sizes of output sequence, which is appropriate to address the problem with a set of network slice requests of different sizes.

- The performance of the proposed EA-NSDP algorithm is verified through extensive simulations. First, different batch sizes and iteration times in the search strategy are implemented to validate the performance. Then, the simulation results demonstrate that our proposed algorithm achieves enhanced performance in terms of energy efficiency, cumulative acceptance ratio and resource utilization compared to two existing algorithms.

The rest of this chapter is organized as follows: Section 5.2 summarizes the related works of the network slicing based on the DRL approaches. Section 5.3 provides the network model, including the physical infrastructure and the network slice requests. Section 5.4 introduces an energy-aware design objective of the NSDP, and gives the details of problem formulation that contains a MDP model with several RL elements. Moreover, the algorithm framework of the proposed EA-NSDP is proposed in Section 5.5, including the structure of the learning agent, the optimization of policy gradient and the details of learning algorithms. In addition, the performance of the proposed algorithms is verified in Section 5.6. Finally, a brief conclusion is summarized in Section 5.7.

5.2 Related Works

The NSDP have been extensively investigated based on different optimization objectives[44, 97, 95]. For example, Reference [44] presented a resource allocation scheme of network slicing, which aimed to minimize the operational expenditures for the infrastructure providers while maximizing the social welfare among different network slice tenants. Bouzidi *et.al* [97] introduced a SDN-based architecture to enable the creation of radio and transport slices through the prediction of slice capacity and congestion, which intended to minimize the overall latency of networks. Besides, Guan *et.al* [95] proposed a service-oriented deployment algorithm for the network slicing to optimize the resource utilization for different kinds of network slices.

Regarding the optimization methods for solving the NSDP, in addition to the two methods (exact-based approaches and heuristic-based algorithms) mentioned above, the DRL technology has been applied to solve the NSDP. Two representative DRL approaches have been widely employed: the value-based Deep Q-learning algorithms [123] and the policy-based Actor-Critic algorithms.

5.2.1 Deep Q-Learning for Network Slicing

Applying the approaches of Deep Q-Learning Networks (DQNs) as a solution for the problems in network slicing has been investigated in recent literature, such as the resource allocation and management problems of network slices and the reconfiguration mechanism.

In Reference [124], the authors demonstrated a demand-aware resource allocation scheme for two network slicing scenarios, including the radio resource slicing and priority-based core network slicing. The allocation scheme adopting the DQNs could incorporate the relationship between user demands and resource supplies, which aimed to enhance the effectiveness and agility for network slices. Suh *et al.* [119] investigated a DRL-based network slicing technique, which attempted to improve the long-term system throughput and satisfy the QoS requirements for different 5G application scenarios. And they employed a DQN to train networks and exploited an action elimination mechanism to speed up training processes by eliminating the undesirable actions. Reference [120] studied a DQN-based resource allocation framework of network slicing, which could be applied in multi-slices and multi-service scenarios. This work was intended to maximize the End-to-End access rate by making dynamic decisions of resource allocation in both RAN slices and core slices. In Reference [125], the authors investigated an intelligent network slicing reconfiguration mechanism under the fluctuation of traffic flows by exploiting the deep reinforcement learning techniques, which aimed to minimize the long-term resource consumption. Specifically, the large state space and high-dimensional discrete action space were decreased by employing the Branching Dueling DQN method. Besides, the work in Reference [126] designed a DQN-based network slicing model in terms of multiple fog nodes collaborating through an edge controller. Regarding the intelligent vehicular applications and smart city scenarios, the proposed DRL-based model attempted to solve the resource allocation problem that required heterogeneous latency and different computing needs.

5.2.2 Actor-Critic for Network Slicing

Although the DQN methods have been investigated widely for network slicing, the Actor-Critic architecture as a technique that combines the strengths of the DQN approaches and the policy gradient mechanisms has been a promising approach for solving the resource allocation and optimization problems in network slicing.

For instance, Reference [127] proposed a resource allocation method in the RAN network slicing in terms of the power and radio resources based on the DRL and DL algorithms considering system power and slice isolation. To be specific, this work employed the Asynchronous Advantage Actor-Critic (A3C) algorithm [86] and SBiLSTM technique to determine the allocation scheme of power and spectrum resources. In Reference [128], a dynamic resource allocation algorithm was developed based on the Proximal Policy Optimization (PPO) method [129] for the network slicing scenarios where heterogeneous requirements were considered in multi-access edge computing environments. Its goal was to maximize the resource efficiency and satisfy the QoS requirements of network slices at the same time by formulating a cooperative multi-agent task. In Reference [121], a resource allocation mechanism for network slicing named DeepSlicing was presented. The DeepSlicing method aimed to firstly learn the resource requirements of users to satisfy their QoS requirements and then optimize the resource management accordingly by integrating the Deep Q-Leaning

Network and Actor-Critic method. In particular, a critic function was implemented to estimate the value function of state-action pairs by using the DQN method. And an actor function performed the mapping actions from a state to a specific action based on the current policy.

However, despite the growing literature on the network slicing problems, there still exists a critical research gap on how to deploy network slices based on DRL for achieving efficient energy utilization while balancing the trade-off between the energy cost and the acceptance ratio of the successful deployments of network slice requests. Therefore, in the following, a novel policy solution to solve the energy-aware network slicing design problem is proposed for deploying multiple network slices onto the physical infrastructure based on heterogeneous requirements, aiming to optimize the energy consumption and enhance the acceptance ratio.

5.3 Network Model of Energy-aware Network Slicing Design Problem

In this section, the energy-aware network models are presented, including the physical infrastructure and network slice requests. The main notations involved in this chapter are summarized in Table 5.1.

5.3.1 Physical Infrastructure Model

A physical network is represented by a weighted undirected graph $G = (P, L, A_P, A_L)$, where P denotes a set of physical nodes, L represents a set of physical paths, and A_P and A_L indicate the attributes sets of P and L respectively. Each physical node p_i is associated with its attribute set $A_{p_i} = (\{c_{p_i}, \kappa_{p_i}, o_{p_i}\} | p_i \in P)$, and $A_{p_i} \subseteq A_P$. Particularly, c_{p_i} stands for the CPU capacity of p_i. And κ_{p_i} is a binary parameter denoting the performing role of p_i in the deployment process. Precisely, $\kappa_{p_i} = 0$ means that p_i is a host node that can host one or more VNFs; otherwise, it is a forwarding node when it does not host any VNFs but virtual links route through it. Besides, each physical node

Table 5.1 Notations of the energy-aware NSDP

Notation	Description
G	Physical infrastructure topology
Q	Network slice topology
$\mathcal{K}$	The number of network slice requests
κ_{p_i}	Performing role of p_i
o_{p_i}	Occupancy state of p_i
a_δ	Energy consumption of per unit of CPU
z_δ	Energy consumption of per unit of bandwidth
$h(\mathcal{L}_{ij})$	Latency of the physical path $\mathcal{L}_{ij}$
K	Batch size
I	Iteration times

p_i is represented by its occupancy state $o_{p_i} = \{0,1\}$. Specifically, $o_{p_i} = 1$ means that p_i is active or powered up; otherwise, it is inactive state.

It is assumed that each unit of CPU capacity of physical nodes consumes a_δ energy to be activated. The physical path between p_i and p_j is defined by $\mathcal{L}_{ij}$, and it is characterized by its attribute set $A_{\mathcal{L}_{ij}} \subseteq A_L$. Specifically, $A_{\mathcal{L}_{ij}}$ is represented by $A_{\mathcal{L}_{ij}} = \{b(\mathcal{L}_{ij}), h(\mathcal{L}_{ij})\}$, where $b(\mathcal{L}_{ij})$ denotes the bandwidth capacity of $\mathcal{L}_{ij}$, and $h(\mathcal{L}_{ij})$ indicates the latency of $\mathcal{L}_{ij}$. In addition, z_δ energy consumption is required to activate one unit of bandwidth capacity.

5.3.2 Network Slice Request Model

Each network slice is associated with an undirected graph $Q = (V,E)$, where V denotes the set of required VNFs and E indicates the set of virtual links. A network slice request q_k is represented by $q_k = (v,e,c_v,b_e|\forall k \in \mathcal{K})$, where v and e stand for the sets of VNFs and virtual links respectively. c_v indicates the set of CPU requirements of v, b_e represents the set of bandwidth requirements of e. And $\mathcal{K}$ denotes the total amount of network slice requests that need to be deployed in the network. Besides, each VNF $v_s \in v$ is associated with its required CPU resources $c_{v_s} \in c_v$. Each virtual link $e_{st} \in e$ is

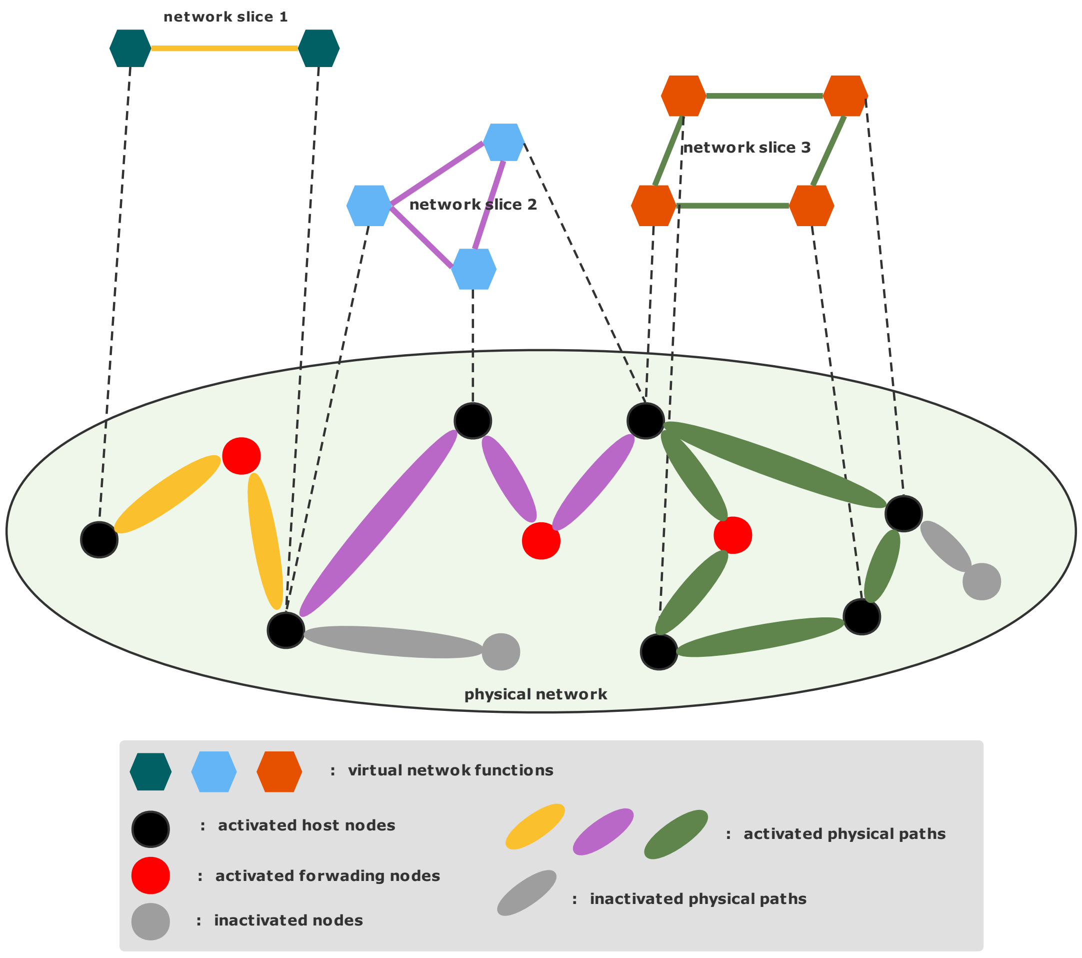

network slice 1
network slice 2
network slice 3
physical network
: virtual netwok functions
: activated host nodes
: activated forwading nodes
: inactivated nodes
: activated physical paths
: inactivated physical paths

Fig. 5.1 Illustration of Energy-aware Network Slicing Design Problem.

a logical connection between the VNFs v_s and v_t, and $b(e_{st}) \in b_e$ denotes the required bandwidth resources of e_{st}.

5.4 Problem Formulation

In this section, firstly, the energy-aware design objective for network slicing is proposed. Then the present EA-NSDP is formulated as a Markov Decision Process (MDP) problem with critical RL elements, including a state space, an action set and a reward function. The illustration of the EA-NSDP processes is shown in Fig. 5.1.

5.4.1 Energy-aware Design Objective

In this chapter, we aim to solve an energy-aware network slicing design problem (EA-NSDP) by deploying various network slice requests onto the physical infrastructure concerning energy consumption. The design objective focuses on optimizing energy consumption to enhance the energy efficiency of the network, including the node energy consumption E_N and link energy consumption E_L, which is defined by:

$$\min \quad \mathscr{E} = E_N + E_L. \tag{5.1}$$

Node Energy Consumption

The node energy consumption of a network slice request q_k can be calculated as follows:

$$E_N(q_k) = \Delta S_k + \Delta D_k + \Delta B_k, \tag{5.2}$$

where ΔS_k denotes the switching power consumption of host nodes for deploying q_k, which indicates the energy cost consumed for transiting energy states from power-saving into activated. ΔD_k indicates the demand power consumption of q_k, which is defined as the amount of power required to host the VNFs in q_k. Besides, ΔB_k represents the basal power consumption with basic CPU load for hosting q_k, specifically, it indicates the amount of power required to maintain host nodes in basic working conditions after being activated.

Let $S_k(p_i)$ denote the boot up or switching power cost for activating a host node p_i from an idle or sleeping mode. The occupancy state of p_i can be represented by:

$$o_{p_i} = \begin{cases} 1, & \sum_{v_s \in v} x_i^s > 0; \\ 0, & \text{otherwise}, \end{cases} \tag{5.3}$$

where $x_i^s \in \{0,1\}$ indicates a binary variable of a VNF deployment. In detail, $x_i^s = 1$ if a physical node p_i hosts a VNF v_s. The value of o_{p_i} is set to 1 when p_i hosts one or more VNFs; otherwise, it is set to 0. The switching power consumption of host nodes ΔS_k for placing q_k is defined by:

$$\Delta S_k = \sum_{p_i \in P_k} o_{p_i} \cdot S_k(p_i), \tag{5.4}$$

where $P_k \in P$ denotes a set of host nodes for deploying q_k.

Further, activating resources on demand can avoid the waste of energy consumption, according to the service requirements of network slices. The demand power consumption ΔD_k can be represented by:

$$\Delta D_k = \sum_{p_i \in P_k} \sum_{v_s \in v} x_i^s \cdot (c_{v_s} a_\delta). \tag{5.5}$$

The basal power consumption ΔB_k can be expressed as:

$$\Delta B_k = \sum_{p_i \in P_k} \sum_{v_s \in v} x_i^s \cdot \rho_{p_i}, \tag{5.6}$$

where ρ_{p_i} represents the amount of power required to maintain basic working condition of a host node p_i, and it is assumed that its value keeps the same for different physical nodes.

The node energy consumption $E_N(q_k)$ of q_k can be calculated as:

$$E_N(q_k) = \sum_{p_i \in P_k} \{o_{p_i} \cdot S_k(p_i) + \sum_{v_s \in v} x_i^s \cdot (c_{v_s} a_\delta + \rho_{p_i})\}. \tag{5.7}$$

Link Energy Consumption

The link energy consumption of q_k can be defined by:

$$E_L(q_k) = \Delta D_k' + \Delta S_k', \tag{5.8}$$

where $\Delta D_k'$ indicates the demand power consumption of a set of physical links, and $\Delta S_k'$ represents the switching power consumption of forwarding nodes in q_k.

Specifically, $\Delta D_k'$ is represented by:

$$\Delta D_k' = \sum_{\mathcal{L}_{ij} \in L_k} \sum_{e_{st} \in e} y_{ij}^{st} \cdot (b(e_{st}) z_\delta) \cdot h(\mathcal{L}_{ij}), \tag{5.9}$$

where L_k represents a set of physical links for placing q_k. And y_{ij}^{st} denotes a binary variable of a virtual link deployment. $y_{ij}^{st} = 1$ indicates that a single virtual link e_{st} resides on a physical path $\mathcal{L}_{ij}$.

Let $S_k(p_\eta)$ denote the boot up power cost to activate a forwarding node p_η for deploying q_k. And $\Delta S_k'$ can be defined by:

$$\Delta S_k' = S_k(p_\eta) \cdot \sum_{p_\eta \in \mathcal{L}_{ij}} \kappa_{p_\eta}. \tag{5.10}$$

The link energy consumption $E_L(q_k)$ of deploying q_k can be represented by:

$$E_L(q_k) = \sum_{\mathcal{L}_{ij} \in L_k} \sum_{e_{st} \in e} y_{ij}^{st} \cdot (b(e_{st}) z_\delta) \cdot h(\mathcal{L}_{ij}) + S_k(p_\eta) \cdot \sum_{p_\eta \in \mathcal{L}_{ij}} \kappa_{p_\eta}. \tag{5.11}$$

The design objective of the EA-NSDP is to minimize the long-term energy consumption of the physical infrastructure G_P. Consequently, it can be defined as follows:

$$\mathscr{E}(G_P) = \lim_{\mathbb{T} \to \infty} \{ \sum_{k \in \mathcal{K}_\mathbb{T}} E_N(q_k) + E_L(q_k) \}, \tag{5.12}$$

where $\mathcal{K}_\mathbb{T} = \{k | 0 < t_k < \mathbb{T}\}$ indicates the set of network slice requests deploying before time $\mathbb{T}$.

5.4.2 MDP Formulation for the Energy-aware NSDP

In RL, a learning agent performs actions, which will cause a transition from the current state to a new state. The agent then will be informed about the goodness of the taken action by receiving a numerical reward provided by the environment based on a reward function. Over time, the agent aims to learn the optimal action at any state and maximize the long-term reward through a set of interactions with the environment.

A RL task can be modeled as a MDP problem when it has Markov property [85]. In the present work, the decision-making processes can be formulated as a MDP problem, because the states in the EA-NSDP are Markov states. To be specific, the EA-NSDP is a VNE-type problem. The state defined in VNE problems is a Markov state when it has Markov property, specifically, when virtual network requests are independent of each other and previous states and actions. The detailed proof can be found in the Proposition 3.1 of Reference [130].

In general, a MDP problem can be formulated by a tuple $(S, A, \mathcal{P}, R, \gamma)$, where S and A indicate a finite state space and a finite action set, respectively. $\mathcal{P}$ denotes the transition probability that a state $s = s_t$ can transfer to a successor state $s' = s_{t+1}$ after an action $a = a_t$ is performed [131]. It can be represented by:

$$\mathcal{P}^a_{ss'} = \mathcal{P}[s' = s_{t+1} | s = s_t, a = a_t]. \tag{5.13}$$

$R_a(s', s)$ represents the reward received after executing the action a in state s leading to state s', which is defined by:

$$R_a(s', s) = \mathbb{E}[r_{t+1} | s = s_t, a = a_t]. \tag{5.14}$$

The objective of a MDP problem is to maximize the long-term reward $\mathcal{G}_t$, which can be calculated by:

$$\mathcal{G}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0} \gamma^i r_{t+i+1}, \tag{5.15}$$

where r_{t+i+1} indicates as an immediate reward. $\gamma \in [0, 1]$ is a discount factor that reflects the decreasing importance of the present reward on futures in the cumulative reward computation.

In this work, the environment consists of the physical infrastructure and a set of network slice requests. The agent corresponds to the Network Slicing Management and Orchestration (NS-MANO) framework, which can generate design solutions for network slice requests after receiving the state of observations from the environment. In addition, the specific definitions of state, action and reward regarding the EA-NSDP are presented in the following.

State Space: The state observation contains the available resource capacities of the physical infrastructure and the resource requirements of network slice requests. The state s_t can be expressed as:

$$s_t = [\mathbf{C}^t, \mathbf{Q}^t, \mathbf{W}^t_{sum}, \mathbf{W}^t_{max}, \mathbf{W}^t_{min}], \tag{5.16}$$

where $\mathbf{C}^t$ represents the available CPU resource capacity of physical nodes P at time step t, it is defined by:

$$\mathbf{C}^t = \{\tilde{c}_{p_1}, \tilde{c}_{p_2}, ..., \tilde{c}_{p_N}\}, \tag{5.17}$$

where N is the total number of physical nodes. $\tilde{c}_{p_i}$ represents the remaining CPU resources of $p_i \in P$. In addition, $\mathbf{Q}^t = \{c_{v_s}, W_{sum}(v_s)\}$ indicates the state of a network slice request q_k, which is defined as the required amount of CPU resource c_{v_s} and bandwidth resource $W_{sum}(v_s)$ of each VNF $v_s \in v$.

Each physical node p_i is associated with a set of adjacent physical links $\mathcal{L}(p_i)$. $\mathbf{W}^t_{sum}$ denotes the set of the sum of available bandwidth resources, which is defined by: $\mathbf{W}^t_{sum} = (W_{sum}(p_1), W_{sum}(p_2), ..., W_{sum}(p_N))$. The available bandwidth resources of $\mathcal{L}(p_i)$ is defined by $W_{sum}(p_i)$, which is shown as follows:

$$W_{sum}(p_i) = \sum_{l_{ij} \in \mathcal{L}(p_i)} \tilde{b}_{l_{ij}} \quad \forall p_j \in P, \tag{5.18}$$

where l_{ij} is an adjacent link of p_i, and $\tilde{b}_{l_{ij}}$ denotes the available bandwidth resources of l_{ij}. It is assumed that p_i can be assigned a higher probability of hosting VNFs when it has more available bandwidth resources. Besides, the set of the maximum of available bandwidth resources is defined by $\mathbf{W}^t_{max} = (W_{max}(p_1), W_{max}(p_2), ..., W_{max}(p_N))$, where $W_{max}(p_i)$ denotes the maximum available bandwidth resource among $\mathcal{L}(p_i)$. Similarly, $\mathbf{W}^t_{min}$ indicates the set of the minimum available bandwidth resources.

Action Space: The action set can be represented as $A = \{a_1, a_2, ..., a_T\}$, which can describe a set of deployment processes of all VNFs in a network slice request q_k. The size of the set of VNFs of q_k is defined as $M_k = T$. To generate a corresponding mapping solution to host a VNF v_t, a certain action $a_t (t \in T)$ can select a candidate physical node whose available resource capacities exceed the requirements of v_t, from the N-dimensional discrete set $P = \{p_1, p_2, ..., p_N\}$. Besides, it is assumed that the previously chosen actions can not be selected again for the same request, which means that the VNFs in the same slice request should be embedded onto different physical nodes.

Reward: The agent can adjust and improve its actions by receiving a correlative reward from the environment, instead of following an explicit objective function in ILP or MILP. It can be guided to deploy network slice requests in the direction of maximizing the expected long-term return based on rewards.

A positive reward will be given to the agent after an effective action is implemented. Specifically, physical nodes are selected, and a set of adjacent virtual links associated with the embedded VNFs are mapped onto the shortest path of the selected physical nodes when resource constraints are satisfied.

In the present work, a reward function can be defined as Eq. (5.12). However, the energy consumption $E_N(q_k)$ of physical nodes can be neglected as it is deterministic. To be specific, it is decided by the three elements in Eq. (5.2), and they are not related to specific deployment decisions of VNFs. Because the demand power consumption ΔD_k is estimated by the requirements of CPU resources, which is deterministic no matter where the VNFs are mapped onto; in other words, the CPU resources will be consumed exactly on demand. Besides, the switching power consumption ΔS_k of host nodes is decided by an exact number of VNFs, and the basal power consumption ΔB_k is evaluated by the same value of the parameter $\rho_{p_i}(p_i \in P)$.

Furthermore, the energy consumption $E_L(q_k)$, in Eq. (5.11), of physical links varies from different actions. The demand power consumption $\Delta D'_k$ of physical links and the switching power consumption $\Delta S'_k$ of forwarding nodes are both decided by the adjacent links of VNFs, and they may vary according to different actions. Therefore, the reward r_t that the agent receives at time step $t = T$ can be simplified as $r_t = -E_L(q_k)$, where $E_L(q_k) = \{\sum_{\mathcal{L}_{ij} \in L_k} \sum_{e_{st} \in e} y_{ij}^{st} \cdot (b(e_{st}) z_\delta) \cdot h(\mathcal{L}_{ij}) + S_k(p_\eta) \cdot \sum_{p_\eta \in \mathcal{L}_{ij}} \kappa_{p_\eta}\}$, and it can be employed to evaluate the quality of actions and guide the agent to make appropriate decisions.

5.5 Algorithm Framework

In this section, we present a DRL-based algorithm framework for solving the energy-aware NSDP based on the theory of Neural Combinatorial Optimization paradigm with Reinforcement Learning [60]. Specifically, the detailed architecture of the learning agent is introduced based on the pointer network structure [61] with the Bahdanau attention mechanism [132]. The proposed framework is trained using the Advantage Actor-Critic (A2C) algorithm [86].

5.5.1 Learning Agent

The model architecture of the learning agent for the EA-NSDP is illustrated in Fig. 5.2, including two RNN models: Encoder and Decoder of Long Short-term Memory (LSTM) cells [133]. The agent aims to generate a set of decisions for deploying network slice requests.

The mechanism used to make decisions by the agent is called a policy $\pi(\cdot|s)$, which is a probability distribution over possible actions under a certain state. Specifically, $\pi(a_t|s_t)$ indicates the probability to perform an action a_t under s_t, which is defined by:

$$\pi(a_t|s_t) = \mathcal{P}[a = a_t|s = s_t]. \tag{5.19}$$

In this work, we employ the pointer network architecture [61] to generate actions. Generally, the architecture is a Sequence-to-Sequence (Seq2Seq) model [134], which is appropriate to solve the problems with a variable size of output sequence. It can select specific pointers from a set of input sequences with an attention mechanism by integrating contributions of different elements of the input sequences, and then the output of the architecture can be formed by a set of indexes to the input.

In our model, two inputs are given to the architecture. To be specific, a set of input sequence of the physical network G_P is represented as a sequence $X = [x_1, x_2, ..., x_N]$. The vector of a physical node p_n is defined as $x_n = \{\bar{p}_n, \tilde{c}_{p_n}, W_{sum}(p_n), W_{max}(p_n), W_{min}(p_n)\} (n \in N)$, where $\bar{p}_n$ is the index of p_n, and $\tilde{c}_{p_n}$ denotes the available CPU resources of p_n. Besides, $W_{sum}(p_n)$ represents the sum of the available bandwidth

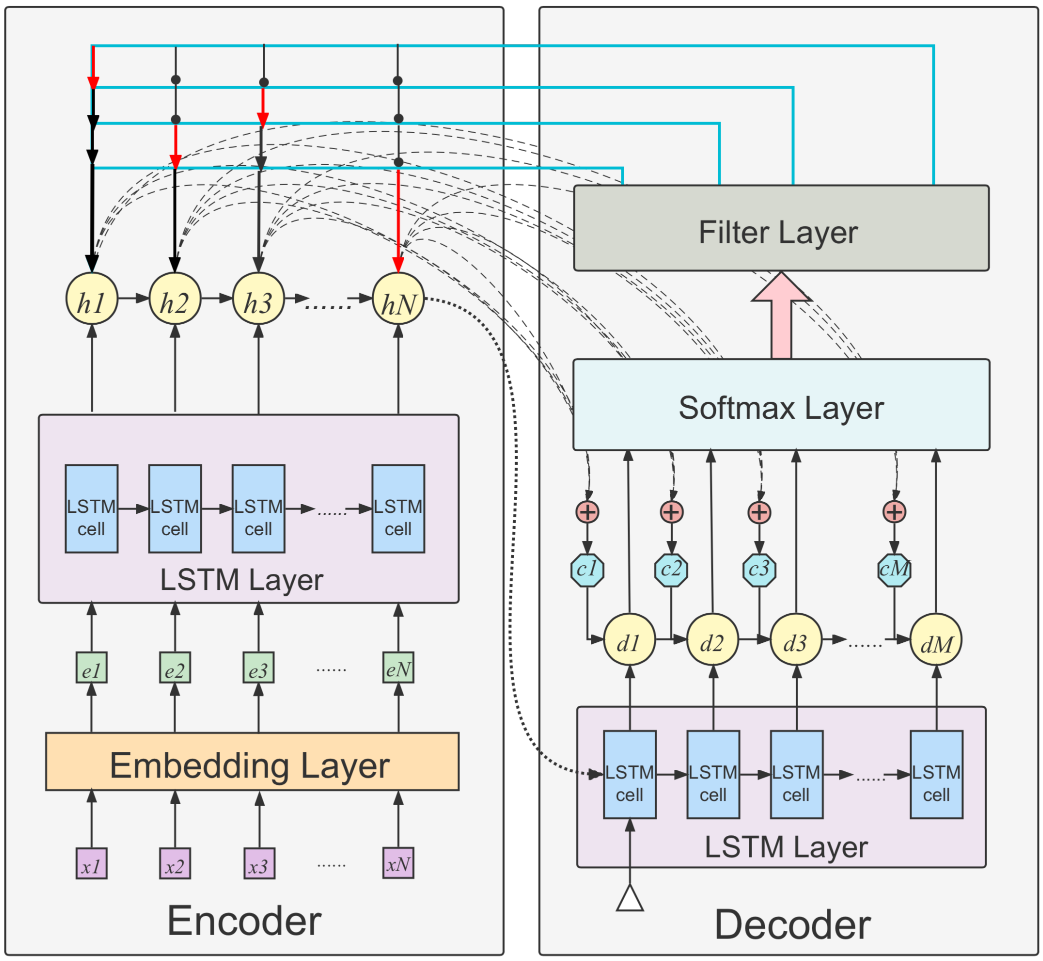


Fig. 5.2 Illustration of the learning agent architecture for the EA-NSDP.

resources of p_n, $W_{min}(p_n)$ is the minimum available bandwidth resources, and $W_{max}(p_n)$ indicates the maximum available bandwidth resources.

Moreover, the input information of a given slice request q_k can be defined as a set of sequences $[\zeta_1, \zeta_2, ..., \zeta_T]$, where $\zeta_s = \{\bar{v}_s, c_{v_s}, W_{sum}(v_s)\}$ denotes the information vector of a VNF v_s. $\bar{v}_s$ represents the index of v_s, c_{v_s} indicates the required CPU resources of v_s, and $W_{sum}(v_s)$ represents the total required bandwidth resources of its adjacent links.

The state information $x_i (i \in N)$ of the physical network is fed into the network model sequentially. An embedding of x_i is denoted as e_i, which is transformed by executing a liner transformation through the embedding layer. The embedding set $\{e_1, e_2, ..., e_N\}$ is inputted into the Encoder network. A set of hidden states $\{h_1, h_2, ..., h_N\}$ can be obtained through the LSTM cell layer. At each Encoder step i, a hidden vector h_i is associated with e_i and the previous hidden vector h_{i-1}. And h_i can be represented by $h_i = f(e_i, h_{i-1})$, where f is a $tanh$ activation function.

The initial input to the Decoder is a Start-of-Sequence (SoS) token denoting as $\triangle$, which is a N-dimensional zero matrix. The initial status of LSTM cells in the Decoder is set as the hidden state of the final step of the Encoder. The hidden states of Decoder network can be expressed as $\{d_1, d_2, ..., d_T\}$. At each time step t, a hidden vector d_t is represented as $d_t = f(d_{t-1}, y_{t-1}, c_t)$, where d_{t-1} indicates the previous hidden vector, y_{t-1} is the output of the previous LSTM cell and c_t is the context vector of the Encoder processes. Besides, c_t can be defined as the sum of hidden states of the input sequence in the Encoder weighted by a set of corresponding attention weights, which is shown as:

$$c_t = \sum_{n=1}^{N} \alpha_{tn} h_n. \tag{5.20}$$

The Decoder network can generate a probability distribution of the design solution of each VNF in q_k over the set of physical nodes using the attention mechanism [132]. Specifically, the Decoder network can pass the information of selected physical nodes of hosting VNFs to the next Decoder step. Let $\mathcal{A}_t = \{\alpha_{t1}, \alpha_{t2}, ..., \alpha_{tN}\}$ represent the attention for selecting physical nodes over the input sequence, and α_{tn} denotes the normalized probability of selecting p_n to the output y_t of the decoding process at the Decoder step t, which is defined by:

$$\alpha_{tn} = \frac{exp(u_{tn})}{\sum_{i=1}^{N} exp(u_{ti})}, \tag{5.21}$$

where the attention score u_{jn} ($\forall n = 1, 2, ..., N$) is defined by:

$$u_{tn} = \begin{cases} -\infty, & \text{if} \quad \tilde{c}_{p_n} < c_{v_t} \quad or \quad W_{sum}(p_n) < W_{sum}(v_t); \\ v_\alpha^\top tanh(W_1 h_n + W_2 d_t), & \text{otherwise}; \end{cases} \tag{5.22}$$

and v_α, W_1 and W_2 are linear learning weights to be learned in the model.

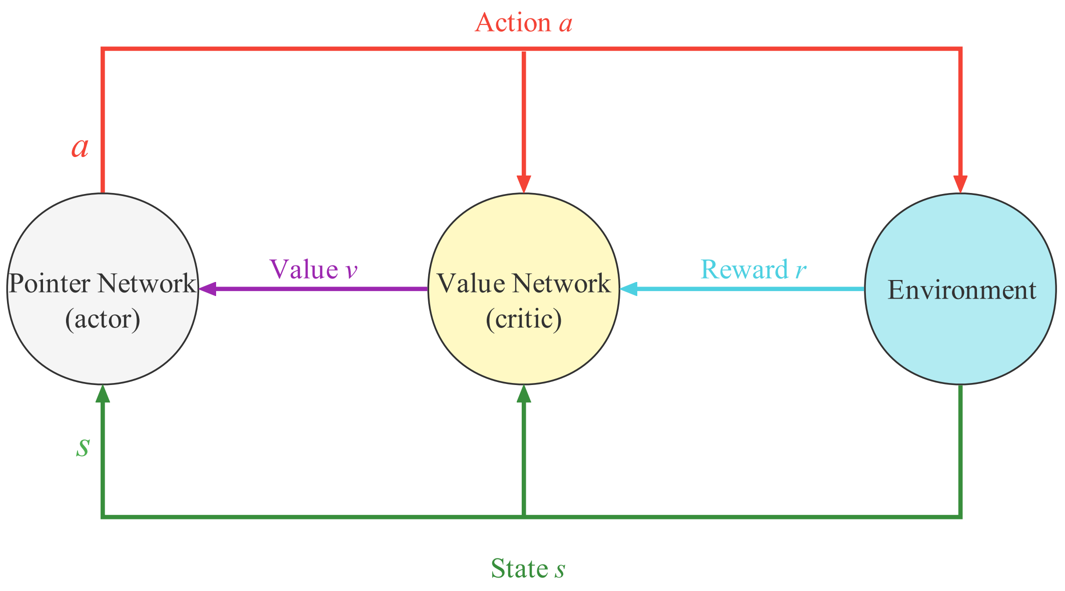


Fig. 5.3 Diagram of the A2C algorithm structure.

Additionally, a filter layer is added to justify the output of the Decoder, aiming to select policies without violation of resource constraints. Its inputs contain the information of both G_P and q_k. A N-dimensional binary vector $\mathcal{M}$ is implemented to guarantee that the selection of physical nodes can satisfy the constraints of network slice mapping. $\mathcal{M}$ is defined as $\mathcal{M} = \{m_1, m_2, ..., m_N\}$, which can be used to select a subset of outputs by filtration. For $\forall v_s \in v$ and $\forall p_n \in P$, m_n is a binary parameter, which can be calculated as follows:

$$m_n = \begin{cases} 0, & \exists c_{v_s} > \tilde{c}_{p_n} \quad or \quad \exists W_{sum}(v_s) > W_{sum}(p_n); \\ 1, & \text{otherwise}; \end{cases} \tag{5.23}$$

where $m_n = 0$ represents that p_n fails to satisfy the resource requirements of the VNF v_s, then the probability of selecting p_n to host v_s will be set to 0. The physical nodes with highest probability will be selected as the output of the Decoder, which is a sampled solution for hosting network slice requests.

5.5.2 Optimization with Policy Gradient

We employ the A2C algorithm architecture [86] to implement learning processes. Generally, the A2C architecture is shown in Fig. 5.3, including two networks: an actor network and a critic network, in which an actor network can be trained under the guidance of a critic network.

Specifically, the actor network represents the present policy network based on the pointer network, which can generate an action a for deploying a VNF according to the state s and value v. The critic network parameterized by θ_v is responsible for estimating the value v of an action sampled by the policy network with parameters θ according to a state-value function. Besides, the actor network and the critic network both update their hidden states by using the attention glimpse mechanism [60] at the memory states and feed the output of the glimpse function as input to the next step of learning process, which is illustrated in Fig. 5.4. The actor network and the critic network can generate different outputs: a policy $\pi_\theta(s_t, a_t)$ and a value estimation $\mathcal{V}_\theta(s_t; \theta_v)$, respectively.

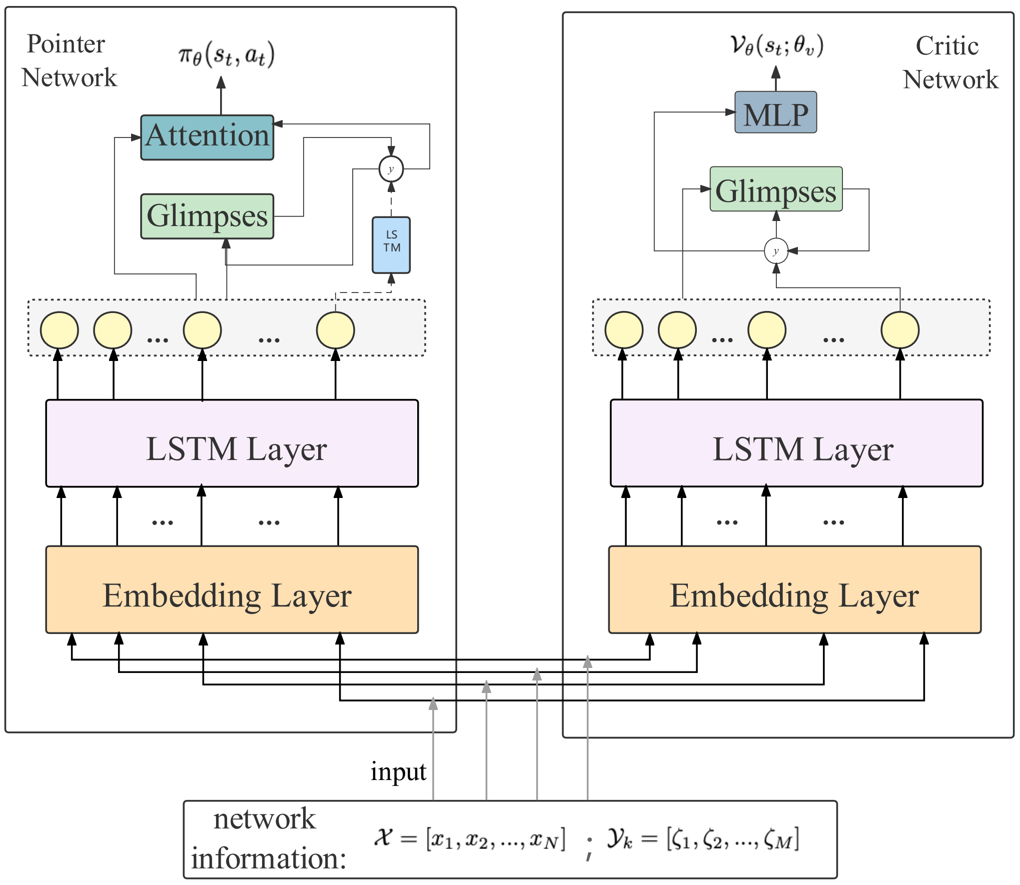


Fig. 5.4 Detailed architecture of the training model for the network slicing design problem.

To be specific, a model-free and policy-based RL method is leveraged to train the policy network by optimizing its parameters θ. At each time step t, for a given physical network input X, the training objective is formulated as:

$$J(\theta|X) = \mathbb{E}_{\pi}[r_t(\tau|X)], \tag{5.24}$$

where $\mathbb{E}[\cdot]$ is an expectation function. $r_t(\tau|X)$ indicates the long-term reward till t of finding sampled trajectory τ with an input sequence X, and τ is defined as $\tau = (s_1, a_1, ..., s_t, a_t)$.

During training processes, the vector of input sequences X is drawn based on uniform distribution from P. We employ the stochastic gradient ascent to optimize the policy network. The parameters θ of policy network can be updated as follows:

$$\theta_{t+1} = \theta_t + \xi \nabla_\theta J(\theta), \tag{5.25}$$

where ξ is a learning parameter. The optimal policy π_{θ^*} can be learned with θ^*, and θ^* is defined by:

$$\theta^* = \mathrm{argmax} \mathbb{E}_{\pi_\theta}[r_t]. \tag{5.26}$$

Differentiating J in terms of θ indicates the direction of the adjustment of network parameters, which aims to update θ to find the global optimal solution efficiently. The gradient $\nabla_\theta J(\theta)$ is formulated based on the policy gradient method introduced in Reference [83], which is expressed as:

$$\nabla_\theta J(\theta) = \underset{\tau \sim \pi_\theta(\tau)}{\mathbb{E}} [(r(\tau|X) - b(X))\nabla_\theta log \pi_\theta(\tau|X)], \tag{5.27}$$

where $b(X)$ is defined as a baseline function to estimate the predicted reward for reducing the variance of the gradients, and it is independent on τ.

The policy gradient $\nabla_\theta J(\theta)$ can be approximated based on the Monte-Carlo sampling by drawing K sampled inputs $X_1, X_2, ..., X_K$ and sampling a design policy for per input $\tau_k \sim \pi_\theta(\cdot|X_k)$. It can be estimated with:

$$\nabla_\theta J(\theta) \approx \frac{1}{K}\sum_{k=1}^{K}(r(\tau_k|\pi_\theta) - b(X_k))\nabla_\theta log\pi_\theta(\tau_k|X_k). \tag{5.28}$$

Further, the learning processes of policy network can be improved with the guidance of a critic network by learning the predicted return value sampled by the current policy π_θ for a given input sequence X_k. Specifically, the critic network can be trained based on the mean squared error between a prediction estimator $\mathcal{B}_{\theta_v}$ and the actual return sampled by the recent policy. The loss function of the critic network $\mathcal{L}(\theta_v)$ can be formulated as:

$$\mathcal{L}(\theta_v) = \frac{1}{K}\sum_{k=1}^{K}[r(\tau_k|X_k) - \mathcal{B}_{\theta_v}(X_k)]^2, \tag{5.29}$$

similar to Reference [60], the baseline estimator $\mathcal{B}_{\theta_v}(X_k)$ is defined as an exponential moving average value of the long-term accumulated reward obtained by the policy network, which can be updated during the learning processes automatically. It can facilitate estimating the expected energy consumption by reducing the variance of the policy gradients and leading to speed convergence.

5.5.3 Leaning Algorithms of Energy-aware Design Policy

In the present work, the training algorithm is based on the A2C architecture since the difference between the sampled return value and the predicted value of the critic network is an unbiased estimation of the advantage function. In A2C, the actor network can sample actions according to the values predicted by the critic network, and the parameters of the actor network can be optimized using the policy gradient mechanism. The critic network improves its evaluation skills based on the reward given by the environment.

The details of the training algorithm is presented in Algorithm 7. The parameters of the actor network and critic network are firstly initialized. Then the policy network with parameters θ can sample a solution τ_k by learning for deploying the network slice request q_k. In the learning processes, the parameters θ are adjusted in the direction of the gradient of the objective function [37], which may tend to converge to local

Algorithm 7 Algorithm of Training Process

```
Input:
    $G_P$: the physical network;
    $q_k$: a network slice request;
    $\mathcal{T}$: the number of training steps;
    $K$: batch size;
1:  initialize the parameters of the policy network $\theta$;
2:  initialize the parameters of the critic network $\theta_v$;
3:  for $1, 2, ..., \mathcal{T}$ do
4:      synchronize parameters $\theta$ and $\theta_v$;
5:      for $k = 1, 2, ..., K$ do
6:          sample an input sequence $X_k$;
7:          sample a solution $\tau_k = (s_1, a_1, ..., s_T, a_T)$;
8:          $b_k \leftarrow \mathcal{B}_{\theta_v}(X_k)$;
9:      end for
10:     $\nabla_\theta J(\theta) \leftarrow \frac{1}{K}\sum_{k=1}^{K}[(r(\tau_k) - b_k)\nabla_\theta log\pi_\theta(\tau_k|X_k)]$;
11:     $\mathcal{L}(\theta_v) \leftarrow \frac{1}{K}\sum_{k=1}^{K}[r(\tau_k) - b_k]^2$;
12:     $\theta \leftarrow$ optimize$(\theta, \nabla_\theta J(\theta))$;
13:     $\theta_v \leftarrow$ optimize$(\theta_v, \nabla_{\theta_v}\mathcal{L}(\theta_v))$;
14: end for
15: return $\theta$.
```

optimums if the objective function is non-convex. In this case, the learning agent can suffer from a local convergence, and the system may fail to learn the optimal policy even though the training process is extended. Besides, each network slice request is hosted by a subgraph of the physical network topology. The number of subgraphs exponentially grows with the increment of infrastructure components. The action space can be extremely large when each possible subgraph is considered, and it can also lead to slow convergence of the learning framework.

Based on this, we employ a search strategy [60] to further improve the efficiency of searching in a large action space. The search strategy can validate candidate solutions sampled by the policy network at the inference process and finally select the optimal one. The details of the search strategy for the EA-NSDP are presented in Algorithm 8, which can update and refine the parameters θ of the policy network and learn how to determine final design solutions. Specifically, the policy network keeps being trained by sampling solutions. Given an input sequence X and the network slice request q_k, candidate design solutions $\tau_1, ..., \tau_K$ are drawn from the policy network $\pi_\theta(\cdot|X, q_k)$.

Algorithm 8 Algorithm of Search Strategy for the EA-NSDP

```
Input:
    $X$: an input sequence of the physical network;
    $q_k$: the information of a network slicing request;
    $I$: the number of iteration times;
    $K$: batch size;
    $\beta$: learning parameter;
Output:
    $\tau^*$: the design solution of the slice request $q_k$;
 1: input the information of $X$ and $q_k$ to the model;
 2: $\tau^* \sim$ initially sample a solution from $\pi_\theta(\cdot|X,q_k)$;
 3: initialize $R_{\tau^*} \leftarrow r(\tau^*|X,q_k)$;
 4: for $i = 1,2,...,I$ do
 5:     for $k = 1,2,...,K$ do
 6:         $\tau_k \sim$ sample a solution based on $\pi_\theta(\cdot|X,q_k)$;
 7:         calculate $r(\tau_k|X,q_k)$ and let $r(\tau_k|X,q_k) = -\infty$ if resource constrains are vio-
            lated;
 8:     end for
 9:     update $\Pi = \{\tau_1,\tau_2,...,\tau_K\}$;
10:     $j =$ argmax $\{r(\tau_1|X,q_k),...,r(\tau_K|X,q_k)\}$;
11:     $R_j \leftarrow r(\tau_j|X,q_k)$;
12:     if $R_j > R_{\tau^*}$ then
13:         $\tau^* \leftarrow \tau_j$;
14:         $R_{\tau^*} \leftarrow R_j$;
15:     end if
16:     $\nabla_\theta J(\theta) \leftarrow \frac{1}{K}\sum_{k=1}^{K}[(r(\tau_k)-b_i)\nabla_\theta log\pi_\theta(\tau_k|X)]$;
17:     $\theta \leftarrow$ optimize$(\theta,\nabla_\theta J(\theta))$;
18:     $b_i \leftarrow \beta b_{i-1} + (1-\beta)\frac{1}{K}\sum_{k=1}^{K} b_k$;
19: end for
20: return $\tau^*$.
```

The learning agent then calculates the reward of each candidate solution sampled from the policy network, and the solution with maximum reward will be selected as the final design solution among them.

During search processes, the baseline estimator b_i is calculated by leveraging the exponential moving average method instead of the critic network in each iteration, since there is no need to differentiate among slice requests. b_i is used to stimulate the agent to select the solutions with better rewards. The parameters of the policy network can be optimized in the direction of the baseline when the reward of a solution is less than the value of baseline estimator. On the contrary, the policy network will explore new findings and optimize its parameters toward new directions. In addition, the algorithm keeps training the policy network and calculating the gradients dynamically in the search processes. Afterwards, the parameters θ of the policy network are further refined and updated automatically.

The testing algorithm is shown in Algorithm 9. In particular, a design solution is failed when the resource constraints of embedding VNFs or virtual links are violated. A design solution of deploying VNFs is feasible if the resource requirements of VNFs in q_k are satisfied by the selected physical nodes, and then its design solution of the corresponding virtual links is calculated based on the Dijkstra shortest path algorithm [135]. Particularly, a design policy is successful only if it consists of both feasible solutions of VNFs and virtual links. In addition, the computational complexity of the proposed algorithm framework is $O(\mathcal{K}(T(1+lgT)+|E|))$. To be specific, for each network slice request $q_k(k \in \mathcal{K})$, the computational complexity is $O(T)$ after the input sequence is fed into the model, and the computational complexity of the virtual link embedding processes is $O(TlgT+|E|)$ when its VNFs are successfully deployed.

The performance of the present algorithm can be evaluated regarding several evaluation criteria, including the energy consumption efficiency, the cumulative acceptance ratio of network slice requests, and the resource utilization of physical nodes and links. Specifically, the cumulative acceptance ratio is defined as the ratio of the accumulated number of the network slice requests successfully deployed to the total number of requests over time. The resource utilization of physical nodes indicates the CPU resource utilization, which is represented by the ratio between the consumed CPU resources and the total CPU capacity of physical nodes in the infrastructure. Besides, the resource utilization of physical links denotes the bandwidth resource utilization.

Moreover, the energy consumption efficiency Δ denotes the ratio of the required energy by network slice requests to the total energy consumption of the network, which is defined by:

Algorithm 9 Algorithm of Testing Process

```
Input:
    $\theta$: the parameters of the policy network;
    $\Gamma$: a set of network slice requests;
Output:
    $solutionSet$
1: initialize a $solutionSet = \phi$
2: for each network slice request $q_k \in \Gamma$ do
3:     sample candidate solutions from the policy network;
4:     select a design solution $\tau^*$ of VNFs for $q_k$ based on Algorithm 2;
5:     if $\tau^*$ is feasible then
6:         place the VNFs of $q_k$ onto the physical network;
7:         obtain the design solution $\varphi^*$ of virtual links of $q_k$ based on the Dijkstra
           Algorithm;
8:         if the design policy $(\tau^* + \varphi^*)$ is feasible then
9:             perform the mapping processes of virtual links;
10:            update resource capacity of the physical network;
11:            $solutionSet$.add($\tau^* + \varphi^*$);
12:            return 'Design Success';
13:        else
14:            release the placed VNFs;
15:            return 'Design Failed';
16:        end if
17:    else
18:        return 'Design Failed';
19:    end if
20: end for
21: return $solutionSet$.
```

$$\begin{aligned}\Delta =[&\frac{\sum_{k\in\mathcal{K}}\{\sum_{p_i\in P}\{\sum_{v_s\in v}(c_{v_s}a_\delta+\rho_{p_i})+o_{p_i}\cdot S_{p_i}\}\}}{\lim_{\mathbb{T}\to\infty}\{\sum_{t_k=0}^{\mathbb{T}}E_N(q_k)+E_L(q_k)\}}\\ &+\frac{\sum_{k\in\mathcal{K}}\{\sum_{l_{ij}\in L}\sum_{e_{st}\in e}b_{e_{st}}z_\delta\}}{\lim_{\mathbb{T}\to\infty}\{\sum_{t_k=0}^{\mathbb{T}}E_N(q_k)+E_L(q_k)\}}].\end{aligned} \tag{5.30}$$

5.6 Performance Evaluation

In this section, we conduct a series of simulations to investigate the performance of the proposed algorithm framework for the EA-NSDP. Besides, a classic heuristic algorithm NodeRank [136] and a RL-based algorithm using Graph-Convolutional

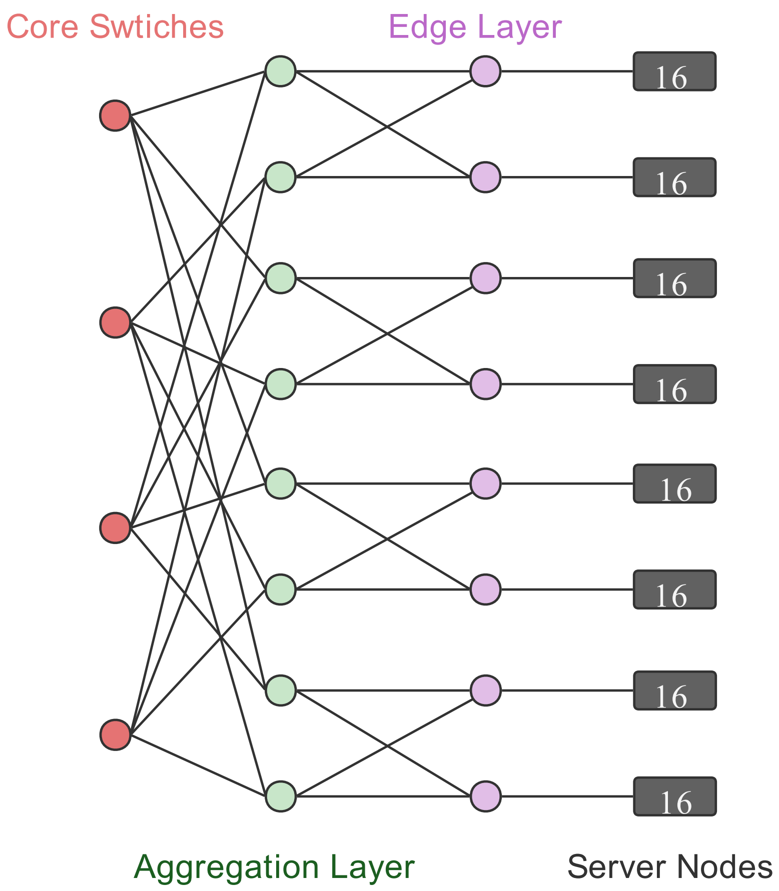


Fig. 5.5 Illustration of the structure of the physical network.

Networks (GCN) [116] are implemented as comparison algorithms to validate the performance of the proposed algorithm.

5.6.1 Simulation Setup

The structure of the physical network is a data centre network with a tree network topology. A similar setting up about the physical network can be found in Reference [137]. The structure of the physical network is illustrated as Fig. 5.5.

Specifically, the physical network consists of 148 physical nodes, including 20 forwarding nodes and 128 host server nodes. The forwarding nodes consists of 4 core switch nodes, 8 aggregation layer nodes and 8 edge layer nodes. The CPU resource capacity of each host server node is set to 10^4. Besides, the physical network has 160 physical links, where there are 128 physical links between the host server nodes and the edge forwarding nodes, and 32 physical links among the other types of forwarding nodes. The bandwidth capacity of the physical links between any host node and the forwarding nodes in the edge layer is given as 10^4, the bandwidth capacity of the physical links between any forwarding node in the edge layer and the forwarding nodes in the aggregation layer is set to $4 \cdot 10^4$, and the bandwidth capacity of the physical links between any core switch node and the forwarding nodes in the aggregation layer is set to $16 \cdot 10^4$.

In addition, the simulations are performed to deploy 500 network slice requests iteratively until all requests are fulfilled. For each network slice request, the number of VNFs is uniformly distributed from 2 to 10. The connectivity of virtual links is set to 0.5, that is to say, the average number of virtual links equals to $(n^2 - n)/4$ and n indicates the number of VNFs of each network slice request. The requirement of CPU resources is uniformly distributed in $U[2 \cdot 10^2, 2 \cdot 10^3]$, and the requirement of bandwidth resources is set to the uniform distribution $U[5 \cdot 10^2, 3 \cdot 10^3]$. And the Time-to-Live (TTL) of each network slice request is uniformly sampled from the distribution $U[10, 10^3]$.

To implement the proposed algorithms for the energy-aware NSDP, the sizes of the LSTM hidden layer and embedding layer are set to 128 respectively. The ADAM optimizer [138] is employed to update the parameters θ of the policy network with a learning rate 10^{-2}, and $\beta_1 = 0.9$, $\beta_2 = 0.999$. Moreover, the discount factor γ is set to 0.9 and the penalty parameter ϕ is adopted with 10^6. The learning parameter β of the baseline estimator is set to 0.9. The parameters of energy consumption a_δ and z_δ are both given as 1. Besides, the switching energy cost of host nodes and forwarding nodes are set to 50 and 10, respectively. The basic energy consumption of host nodes is set to 50, and the maximum time units of the simulation is 10^3. The performance of the proposed algorithms are evaluated under different batch sizes K and iteration times I. Specifically, the value of K is respectively set to $8, 16, 32, 64, 128, 256, 512$. And the value of I is given as $10, 20, 30, 40, 50, 60, 70$, respectively.

Additionally, two comparison algorithms using the NodeRank algorithm and the GCN architecture are validated with the same set of network slice requests to evaluate the effectiveness of the proposed model. For the GCN model, we use the same training

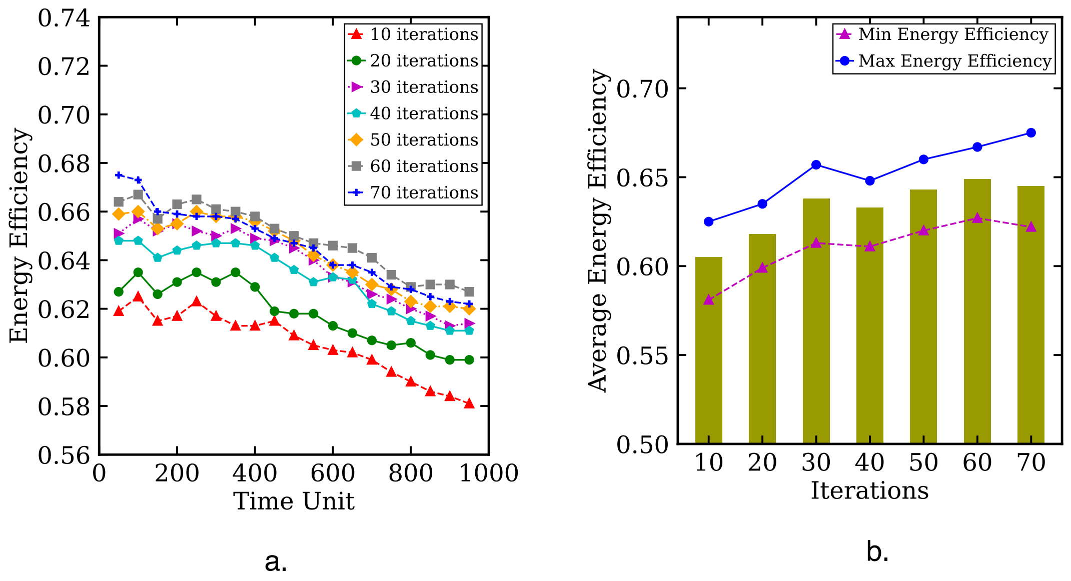


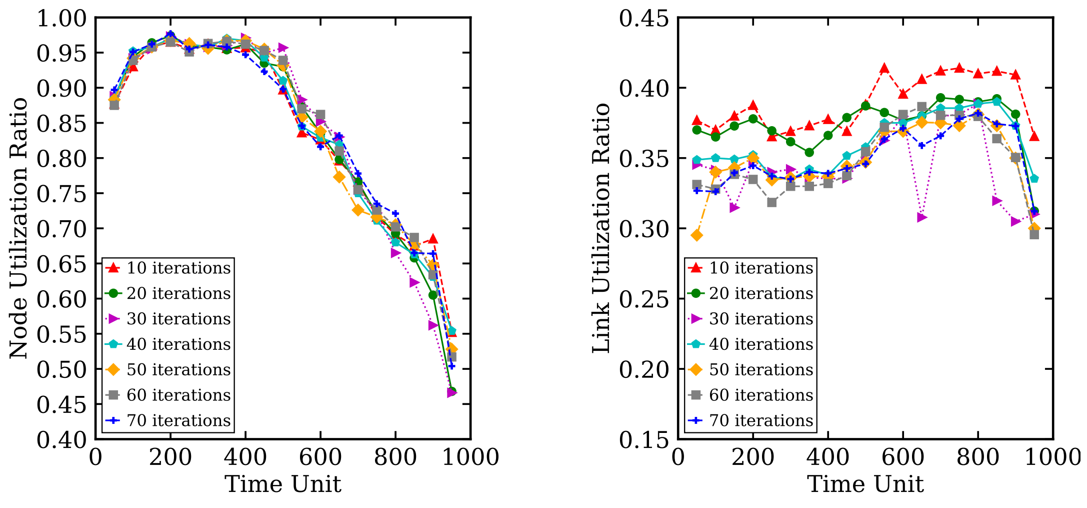
Fig. 5.6 Energy efficiency with different iteration times of the batch size $K = 32$: a. Energy efficiency under different time unit; b. Average energy efficiency for 1000 time units.

parameters as those of the proposed model, specifically, its hidden units are also set to 128, and the learning rate and discount factor are given as the same value 10^{-2} and 0.9, respectively.

The simulations in the present work are executed on a laptop with an Intel Core i7 CPU at 2.50 GHz with 32 GB of RAM. And the models are trained on a NVIDIA RTX A3000 GPU with CUDA 10.1 to enhance the computational efficiency. The DRL environments are built with Python 3.7 and PyTorch 1.6.

5.6.2 Evaluations Results

Firstly, the performance is investigated under different iteration times with the same value of $K = 32$. The energy efficiency changing over time units is shown in Fig. 5.6-a under different iteration times, where it is reduced as the time units increase since more energy is consumed over time. Besides, the energy efficiency rises as the iteration times increase. Because the model can be optimized more thoroughly over larger iteration times and the optimal policy with maximum reward has a great chance



a. b.

Fig. 5.7 Resource utilization ratios with different iteration times of the batch size $K = 32$: a. Node utilization ratio under different time unit; b. Link utilization ratio under different time unit.

of being selected. The average energy efficiency of each iteration time for 1000 time units is also illustrated in Fig. 5.6-b.

Furthermore, the utilization ratios of physical resources are shown in Fig. 5.7 under different iteration times with $K = 32$. Fig. 5.7-a shows that the node utilization ratio is enhanced over time before the time unit of 200 due to the increment of the number of physical nodes hosting VNFs. Afterwards, it gradually decreases with the increase of time units because the use of CPU resources is less than that of release over time. Besides, the node utilization ratio for different iteration times has a similar trend since the consumed CPU resources are the amount of the resources required by the network slice requests. Additionally, in Fig. 5.7-b, the link utilization ratio gradually decreases with the increase of iteration times from the overall trends. The shortest physical paths can be more likely to be found as the quality of design policies enhances when the iteration times are increased. The calculation of the energy efficiency is related to the link occupancy, and it can be improved with the efficient utilization of physical link resources.

Fig. 5.8-a shows the execution time of deploying a network slice request with different iteration times of the batch size of $K = 32$. Obviously, the execution time

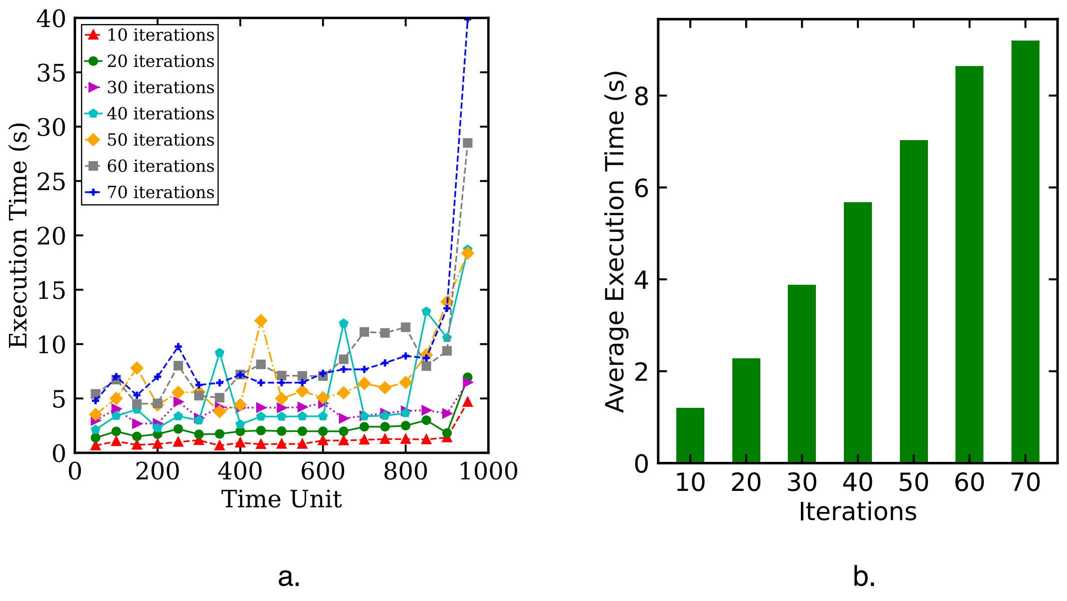


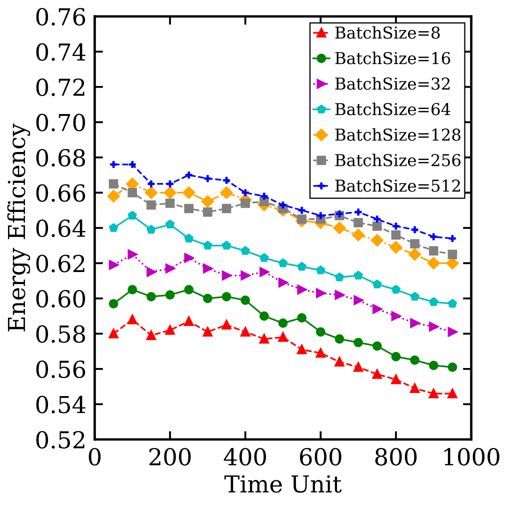
Fig. 5.8 Execution time with different iteration times of the batch size $K = 32$: a. Execution time of deploying a network slice request under different time unit; b. Average execution time for 1000 time units.

generally rises with the increment of the iterations even though some exceptions exist. It gradually increases over time units since the network slice requests that failed to be deployed will try again after the physical resources occupied by the expired requests are released. In addition, the network slice requests with a relatively large number of VNFs may have more complex topological attributes and need more time to be deployed, they are more likely to be deployed successfully only when the physical network is idle after resources are released. The average execution time with different iteration times is shown in Fig. 5.8-b. As can be seen that the average execution time under the iteration time of 70 is more 6% than that under the iteration time of 60, however, its energy efficiency is not as good as that of 60 from Fig. 5.6. Thus, the iteration time is selected as 60 to improve the energy efficiency.

Moreover, the performance can be validated under different batch sizes. From Fig. 5.8, it can be seen that the execution time is shortest when the iteration time I equals to 10. To save the simulation time, we set I as 10 when the differences among various batch sizes are investigated. Specifically, from Fig. 5.9-a, the energy efficiency increases as the increment of batch sizes when K is less than 128. Generally, within a specific range of batch size, the quality of solutions is increased as the batch size increases since the ascending direction of the policy gradient is more accurate and its oscillation is minor. And a small batch usually introduces more randomness, which may make achieving convergence to better solutions more difficult. In addition, the average energy efficiency for 1000 time units is illustrated in Fig. 5.9-b, where the energy efficiency remains in a similar range when the batch size K is set to 128 and 256, respectively, at $I = 10$. To further investigate their differences, the energy efficiencies of $K = 128$ and $K = 256$ are evaluated again when the iteration times I are set to 60, which is shown in Fig. 5.10. As can be found that the energy consumption of $K = 256$ is slightly more efficient than that of $K = 128$.

Fig. 5.11 shows the resource utilization of physical links with different batch sizes under $I = 10$. We can see that less link resources are consumed as batch sizes increase, which makes the deployment of network slice requests more efficient. Because small batch sizes may make the loss function oscillate and cause it a slow convergence to optimal solutions. Besides, it can be seen that the link utilization ratios with the batch sizes of 256 and 512 are similar. Regarding the execution time of deploying a network slice request, it is shown in Fig. 5.12-a when the iteration times equal to 10 for different batch sizes. The execution time increases with the increment of batch sizes. In particular, the execution time of the batch size $K = 512$ is dramatically increased after the time unit of 900. In Fig. 5.12-b, the average execution time for 1000 time units when $K = 512$ is 2.81 times as large as that of the batch size $K = 256$. However, from Figs. 5.9-a and 5.11, it can be seen that the results of the energy efficiency and the link utilization ratio are not enhanced significantly when the batch size is set to 512. Hence, the batch size is chosen as 256 to obtain a relatively efficient energy consumption and enhance the execution efficiency under trade-offs.

Furthermore, the performance of the EA-NSDP can be verified by comparing that of the GCN and NodeRank algorithms with respect to the cumulative acceptance ratio, the energy efficiency and resource utilization of physical nodes and links, when the batch size is set to 256 and the iteration time is given as 60. Regarding the acceptance ratio in Fig. 5.13, it can be observed that the algorithm of the EA-NSDP performs better than the other two comparison algorithms. For example, at 800 time unit, the



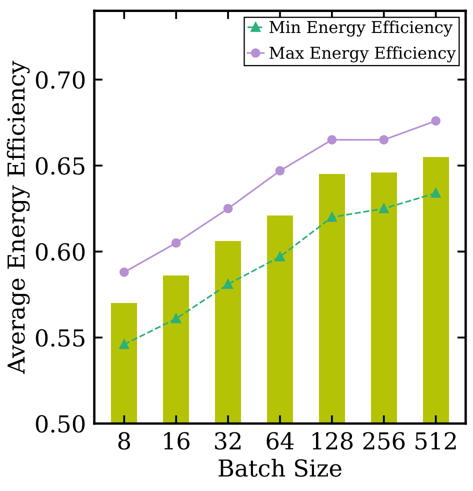


Fig. 5.9 Energy efficiency with different batch sizes of the iteration time $I = 10$: a. Energy efficiency under different time unit; b. Average energy efficiency for 1000 time units.

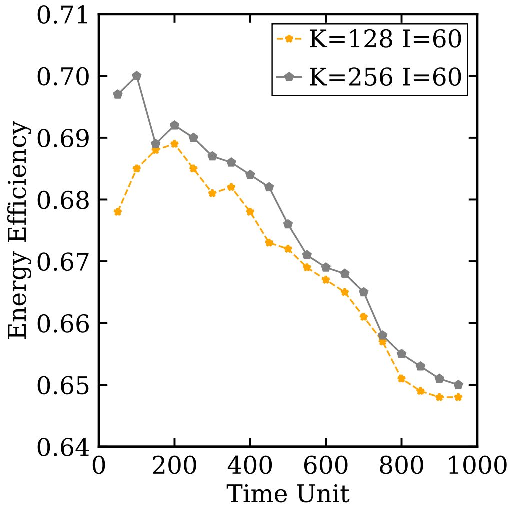


Fig. 5.10 Energy efficiency with the batch sizes $K = 128, 256$ of the iteration time $I = 60$.

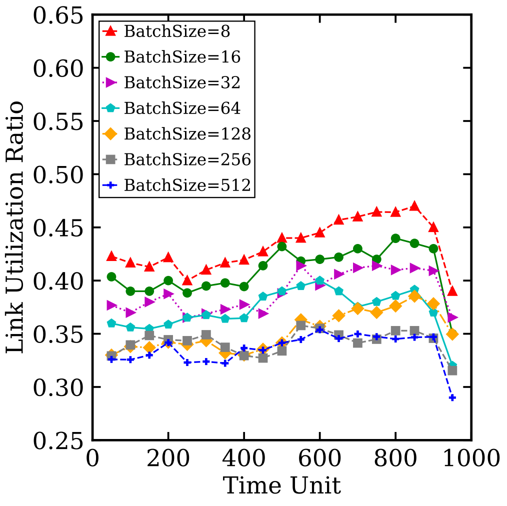
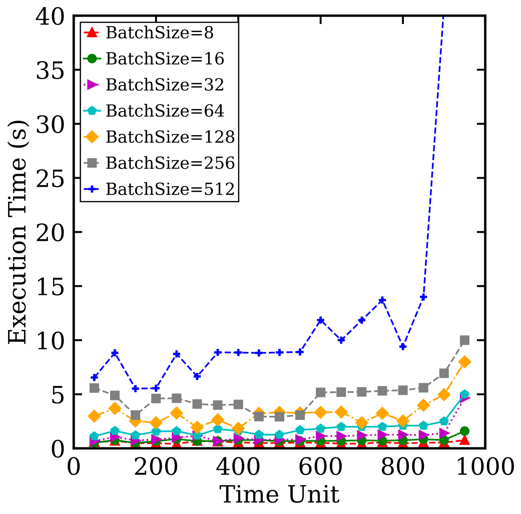


Fig. 5.11 Resource utilization ratio of the physical link with different batch sizes of the iteration time $I = 10$ under different time unit.



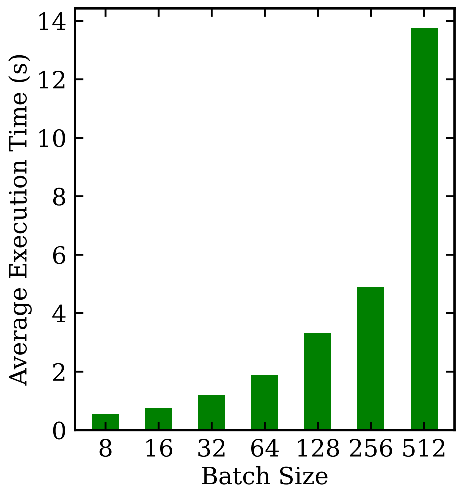


Fig. 5.12 Execution time with different batch sizes of the iteration time $I = 10$: a. Execution time of deploying a network slice request under different time unit; b. Average execution time for 1000 time units.

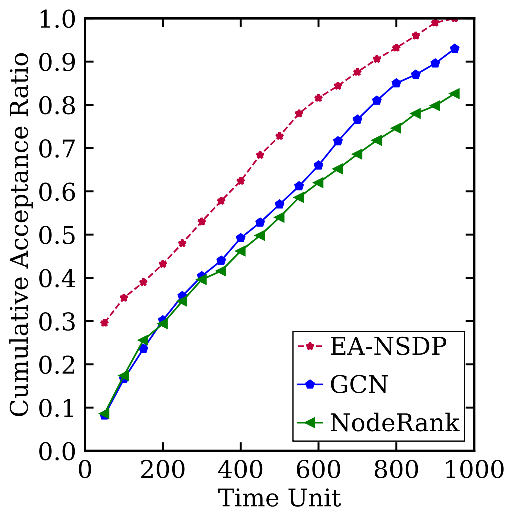


Fig. 5.13 Cumulative acceptance ratio: a comparison between the proposed algorithms of the EA-NSDP and two exiting algorithms GCN and NodeRank.

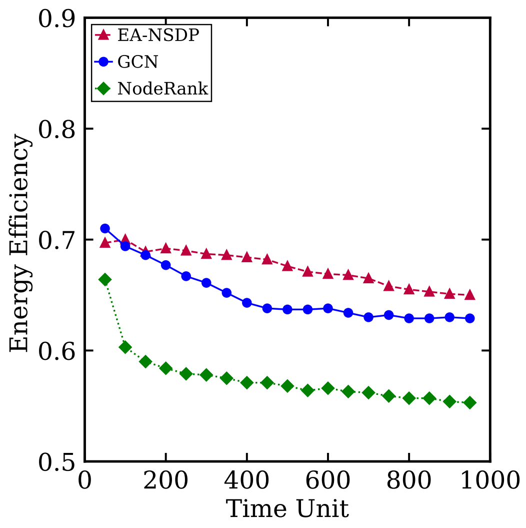


Fig. 5.14 Energy efficiency: a comparison between the proposed algorithms of the EA-NSDP and two exiting algorithms GCN and NodeRank.

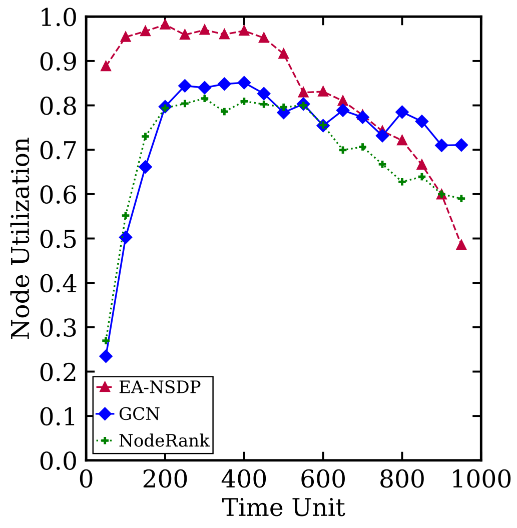


Fig. 5.15 Resource utilization ratio: a comparison between the proposed algorithms of the EA-NSDP and two exiting algorithms GCN and NodeRank.

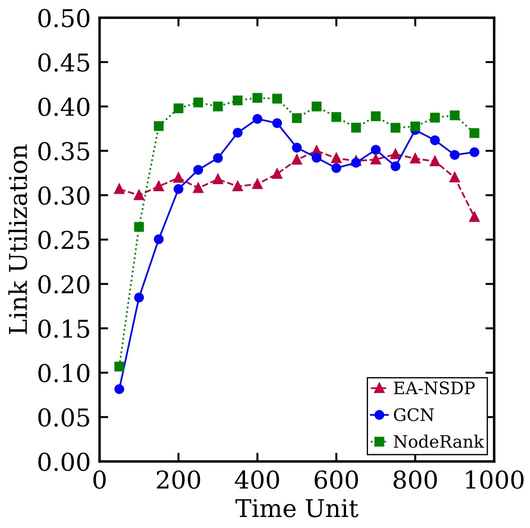


Fig. 5.16 Resource utilization ratio: a comparison between the proposed algorithms of the EA-NSDP and two exiting algorithms GCN and NodeRank.

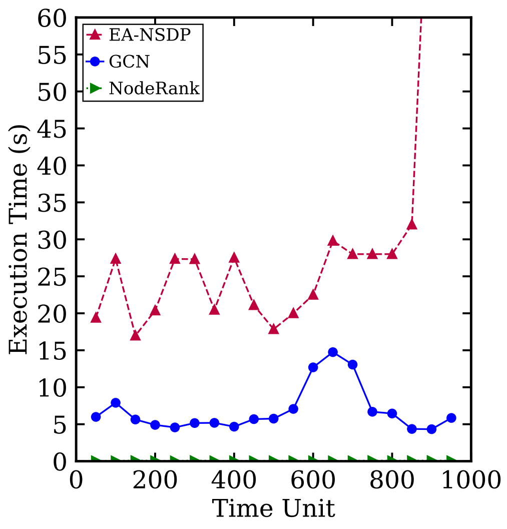


Fig. 5.17 Execution time: a comparison between the proposed algorithms of the EA-NSDP and two exiting algorithms GCN and NodeRank.

cumulative acceptance ratio of the proposed algorithm is more 9.64% and 24.93% than that of the GCN and NodeRank algorithms, respectively. Because its candidate solutions are first sampled by the pointer network of a LSTM layer with the attention mechanism, which leverages the historical experience to guide the policy network to find the feasible solutions. On this basis, the near optimal design policy with maximum reward can be obtained through an effective search strategy, which increases the probability of successful mappings of slice requests. Besides, the cumulative acceptance rate of network slice requests continues to increase over time. The reason is that more slice requests are mapped successfully onto the physical network over time even though some of them once failed to be deployed in the early stage.

Fig. 5.14 illustrates the impact of different algorithms on the energy consumption of mapping network slice requests. Generally, the performance of algorithms based on RL approaches is better than that based on heuristic algorithms. Heuristic algorithms usually lack a holistic consideration of global characteristics of the physical network, which may cause the failure and inefficiency of VNFs mapping. Specifically, the energy consumption of the EA-NSDP and GCN algorithms is more efficient than that of the NodeRank algorithm. Because the NodeRank algorithm cannot learn the characteristics of the physical network by an intelligent agent as RL-based algorithms, and it only mapped the VNFs according to the rank of local importance of the attributes of physical nodes. Besides, the performance of the EA-NSDP algorithm is better than that of the GCN algorithm in terms of the energy efficiency. The GCN algorithm failed to leverage the previous experience to search for candidate solutions despite that it considered the attention mechanism to map VNFs by computing the importance of the adjacent nodes to a certain physical node.

In Fig. 5.15, the resource utilization of physical nodes is investigated compared to the GCN and NodeRank algorithms. In the early stage, the resource utilization of physical nodes increases over time since as many network slice requests are tried to deploy onto the physical network, afterwards, it decreases gently when the resources occupied by the VNFs in the expired network slice requests are gradually released. In addition, it can be seen that the EA-NSDP algorithm performs better than the other two algorithms most of the time in terms of the resource utilization of physical nodes. The trend of the resource utilization of physical links remains correspondingly flat in Fig. 5.16. Because the released resources and the reallocated resources of physical links keep relatively balanced. The resource utilization of physical links of the EA-NSDP algorithm is lower than that of the other two since the proposed algorithm can search the solution with shortest physical paths from the candidate ones. The average execution time of the three algorithms for 1000 time units is shown in Fig. 5.17. The overall execution time of the EA-NSDP algorithm is relatively longer than the other two algorithms since it takes more time to sample candidate solutions by the policy network and determine the final solution by a set of search processes. In addition, it can be seen that the heuristic NodeRank algorithm requires the least amount of execution time.

5.7 Conclusions

In this chapter, we present an energy-aware network slicing design problem to improve the energy efficiency of the deployment of network slices while improving the acceptance ratio. Specifically, an energy-aware objective function of the problem is designed in terms of the energy consumption of physical nodes and links. The proposed problem is formulated as a MDP problem with a reward function considering the link energy consumption. Moreover, the learning network is trained by leveraging the A2C algorithm architecture, where the parameterized policy network is regarded as an actor to sample candidate solutions under the guidance of a critic network. In particular, the policy network is modeled based on the pointer network architecture with the attention mechanism. The parameters of the policy network are optimized using the policy gradient mechanism. A search strategy is presented to update the policy network and determine the final design solution. Besides, compared to two existing RL-based and heuristic approaches, the simulation results show that the proposed algorithm performs better in terms of energy efficiency and cumulative acceptance ratio. We found that the energy efficiency of the proposed algorithm can be enhanced to 69.7%, while the accumulative acceptance ratio can be achieved at 99%. Overall, the present work may further pave the way of optimizing the energy efficiency of network slices using DRL in 5G or beyond technology.

In the next chapter, a comprehensive conclusion of the thesis and a brief introduction to future works are stated.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

In the present thesis, the network slicing design problem is proposed and stated in Chapter 1, and the motivation for investigating the problem is also introduced. The problem is studied with different design objectives for deploying various network slices onto a shared physical infrastructure, which can satisfy diverse service requirements of different use cases and enhance resource and energy efficiency.

Chapter 2 gives a comprehensive introduction to the technical background of network slicing research. Specifically, the fundamental concepts in the End-to-End network slicing technology are introduced, including the virtualization technique, the differences between the virtual machines and containers, the basic knowledge of the Network Function Virtualization and Software Defined Networking techniques, the cloud and edge computing, and the isolation issues in network slices. The principles for implementing network slices are also stated, and the overall framework of network slices is introduced. Besides, the detailed characteristics of different service application scenarios are given, especially the 5G use cases: eMBB, uRLLC and mMTC. In addition, the proposed network slicing design problem is derived from two prerequisite techniques: the Virtual Network Embedding and the Service Function Chaining Placement. And the two techniques are introduced. Moreover, two critical enabling technologies for solving the network slicing design problem are illustrated, including the optimization methods and the deep reinforcement learning approaches.

In Chapter 3, a basic design policy of network slicing is proposed to deploy different network slices in various 5G use cases, aiming to guarantee them coexist in the same physical network. Firstly, the network slicing design problem is modeled as an ILP problem with multiple objectives for eMBB, uRLLC and mMTC network slices under the constraints of limited resources. And a heuristic algorithm is proposed to solve the problem as a trade-off between the computational cost and the quality of solutions since the formulated problem is a NP-hard problem. As a result, the average occupancy ratios of the physical nodes and links illustrate that the proposed algorithm can ensure a balanced utilization of the physical network. With the help particle iterations, the resource efficiency of the proposed algorithm can achieve better performance compared to existing approaches.

Moreover, different service requirements of network slices usually need to be satisfied in various application scenarios. The traffic demands in the network may fluctuate or burst over a period of time. Thus, in Chapter 4, a service-aware design policy is proposed for various use cases, which can deploy network slices and satisfy their resource requirements under the situation that the traffic demands fluctuate in the network. The proposed service-aware problem is formulated as a robust optimization model with the uncertain traffic demands, which allows network slices to provide robust communication services. Using the ILP, the present optimization model can be solved for small scale networks, and the impact on the objective function is verified in terms of different values of robustness coefficients. Besides, a heuristic algorithm is proposed to achieve efficient computation, and the simulation results show that the present algorithms perform better in resource efficiency than the existing algorithm.

In addition, the energy consumption in the network has not been thoroughly considered during the deployment processes of network slices. Therefore, in Chapter 5, an energy-aware design policy is studied to guarantee efficient energy consumption while improving the number of network slices deployed onto the physical network. Specifically, an energy-aware objective is designed according to the energy consumption of the physical nodes and links. The proposed energy-aware problem is modeled as a MDP problem with a reward function of the energy consumption of physical network links. Further, the problem is solved by proposing a design policy based on the DRL approach. The learning network is trained by leveraging the Advantaged Actor-Critic algorithm. The policy network is built as an actor network to sample design policies based on the pointer network architecture of RNNs structure with an attention mechanism, and its parameters are optimized using the policy gradient method. A search strategy is proposed to generate final design solutions and refine the parameters of the policy network. The simulation results validate that the performance of the proposed algorithms is better than those of two exiting algorithms in terms of the energy efficiency and cumulative acceptance ratio.

6.2 Future Works

Although this thesis covers some critical issues in the network slicing design problem, there are still significant issues that have not been considered in the present thesis. In the section, some insights and research directions for future works are stated.

5G networks are usually dynamic, which can lead to varying service requirements. Thus, it is necessary to solve the network slicing design problem in dynamic scenarios. However, the proposed design policies in this thesis are static, which are not effective to deploy network slices dynamically.

One research direction would be investigating a reconfiguration scheme of network slices. A reconfiguration scheme can migrate the previously deployed network slices and reallocate the configured resources. It can utilize newly added resources for accepting more network slices, and the resources can be utilized more efficiently compared to static design policies. It can also greatly simplify the design, management and deployment process of network slices and enhance the flexibility and adaptation of the system for supporting services in dynamic situations [96]. Thus, the design policies presented in Chapters 3, 4 and 5 can be further improved when network slices can be reconfigured according to the variations in service requirements.

Another research direction would be the study of a design policy of network slicing based on a prediction mechanism. In this thesis, the network resources of different network slices are allocated statically according to an expected amount without considering the future of network slice requests and their resource requirements. Besides, the fluctuations of traffic demands in Chapter 4 are modeled based on a simple prediction mechanism, ignoring the prediction of future variations according to the previous resource allocations. Thus, the impact of deploying a specific network slice on future network slice requests and the prediction of resource requirements based on historical experience have not been verified in a dynamic environment. Generally, a prediction mechanism can be extracted from a large amount of historical data. The machine learning technologies are appropriate to be applied to exploit the principles behind raw data and derive predictions of new network slice requests. To sum up, it will be helpful to design policies of network slicing by investigating a prediction mechanism of future network slice requests, for instance, the work proposed in [99].

Additionally, it is worth exploring a feasible security mechanism in the design policy of network slicing. The security challenges for network slicing mainly contain inter-slice security and intra-slice security. To be specific, the inter-slice security refers to the security issues related to other network slices, and intra-slice security is focused on the security aspects of a network slice by itself [139]. An inter-security problem can occur when a dedicated device is only authorized to access a specific network slice but tries to obtain access to other unauthorized ones. Besides, an intra-security issue may damage a network slice by attacking the service running on it. The more significant number of virtual network functions shared by different network slices may lead to a higher security vulnerability in a specific network slice. Thus, a proper isolation level for network slices is essential to carry out their implementations, which can provide a protection mechanism to defend individual network slices. Isolation of network slices can be performed in two ways: physical isolation and virtual machine-based isolation, which consists of different attributes, such as VNFs isolation, traffic isolation, processing isolation and storage isolation. Providing a correct and required isolation level for deploying network slices is one of the most difficult challenges for the design policy of network slicing. Thus, it is a topic of significant research value.

References

[1] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[2] Shunliang Zhang. An overview of network slicing for 5g. *IEEE Wireless Communications*, 26(3):111–117, 2019.

[3] Hassan Hawilo, Abdallah Shami, Maysam Mirahmadi, and Rasool Asal. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE network*, 28(6):18–26, 2014.

[4] Jose Ordonez-Lucena, Pablo Ameigeiras, Diego Lopez, Juan J Ramos-Munoz, Javier Lorca, and Jesus Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges. *IEEE Communications Magazine*, 55(5):80–87, 2017.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[6] Spyridon Vassilaras, Lazaros Gkatzikis, Nikolaos Liakopoulos, Ioannis N Stiakogiannakis, Meiyu Qi, Lei Shi, Liu Liu, Merouane Debbah, and Georgios S Paschos. The algorithmic aspects of network slicing. *IEEE Communications Magazine*, 55(8):112–119, 2017.

[7] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[8] NGMN Alliance. Description of network slicing concept. *NGMN 5G P*, 1(1), 2016.

[9] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[10] Muntasir Raihan Rahman and Raouf Boutaba. Svne: Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management*, 10(2):105–118, 2013.

[11] Marouen Mechtri, Chaima Ghribi, Oussama Soualah, and Djamal Zeghlache. Nfv orchestration framework addressing sfc challenges. *IEEE Communications Magazine*, 55(6):16–23, 2017.

[12] Marouen Mechtri, Chaima Ghribi, and Djamal Zeghlache. A scalable algorithm for the placement of service function chains. *IEEE transactions on network and service management*, 13(3):533–546, 2016.

[13] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[14] Matias Richart, Javier Baliosian, Joan Serrat, and Juan-Luis Gorricho. Resource slicing in virtual wireless networks: A survey. *IEEE Transactions on Network and Service Management*, 13(3):462–476, 2016.

[15] David Marshall. Understanding full virtualization, paravirtualization, and hardware assist. *VMWare White Paper*, 17:725, 2007.

[16] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, volume 3, pages 191–206. Citeseer, 2003.

[17] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[18] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

[20] Carl A Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[21] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: a performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, pages 386–393. IEEE, 2015.

[22] Mehmet Ersue. Etsi nfv management and orchestration-an overview. *Presentation at the IETF*, 88, 2013.

[23] Wei Yang and Carol Fung. A survey on security in network functions virtualization. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 15–19. IEEE, 2016.

[24] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[25] Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent software-defined network updates. *IEEE Communications Surveys & Tutorials*, 21(2):1435–1461, 2018.

[26] Xin Li, Mohammed Samaka, H Anthony Chan, Deval Bhamare, Lav Gupta, Chengcheng Guo, and Raj Jain. Network slicing for 5g: Challenges and opportunities. *IEEE Internet Computing*, 21(5):20–27, 2017.

[27] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[28] Alisa Devlic, Ali Hamidian, Deng Liang, Mats Eriksson, Antonio Consoli, and Jonas Lundstedt. Nesmo: Network slicing management and orchestration framework. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1202–1208. IEEE, 2017.

[29] Amal Kammoun, Nabil Tabbane, Gladys Diaz, Abdulhalim Dandoush, and Nadjib Achir. End-to-end efficient heuristic algorithm for 5g network slicing. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 386–392. IEEE, 2018.

[30] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[31] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on networking*, 20(1):206–219, 2011.

[32] Xiang Cheng, Sen Su, Zhongbao Zhang, Kai Shuang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology awareness and optimization. *Computer Networks*, 56(6):1797–1813, 2012.

[33] Haipeng Yao, Sihan Ma, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Song Guo. A continuous-decision virtual network embedding scheme relying on reinforcement learning. *IEEE Transactions on Network and Service Management*, 17(2):864–875, 2020.

[34] Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li. Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1040–1057, 2020.

[35] Shidong Zhang, Chao Wang, Junsan Zhang, Youxiang Duan, Xinhong You, and Peiying Zhang. Network resource allocation strategy based on deep reinforcement learning. *IEEE Open Journal of the Computer Society*, 1:86–94, 2020.

[36] Mahdi Dolati, Seyedeh Bahereh Hassanpour, Majid Ghaderi, and Ahmad Khonsari. Deepvine: Virtual network embedding with deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 879–885. IEEE, 2019.

[37] Ruben Solozabal, Josu Ceberio, Aitor Sanchoyerto, Luis Zabala, Bego Blanco, and Fidel Liberal. Virtual network function placement optimization with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):292–303, 2019.

[38] Yansen Xu and Ved P Kafle. An availability-enhanced service function chain placement scheme in network function virtualization. *Journal of Sensor and Actuator Networks*, 8(2):34, 2019.

[39] Meng Wang, Bo Cheng, and Junliang Chen. An efficient service function chaining placement algorithm in mobile edge computing. *ACM Transactions on Internet Technology (TOIT)*, 20(4):1–21, 2020.

[40] Insun Jang, Dongeun Suh, Sangheon Pack, and György Dán. Joint optimization of service function placement and flow distribution for service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2532–2541, 2017.

[41] Mohammad Ali Khoshkholghi, Michel Gokan Khan, Kyoomars Alizadeh Noghani, Javid Taheri, Deval Bhamare, Andreas Kassler, Zhengzhe Xiang, Shuiguang Deng, and Xiaoxian Yang. Service function chain placement

for joint cost and latency optimization. *Mobile Networks and Applications*, 25(6):2191–2205, 2020.

[42] Marwa A Abdelaal, Gamal A Ebrahim, and Wagdy R Anis. Efficient placement of service function chains in cloud computing environments. *Electronics*, 10(3):323, 2021.

[43] Racha Gouareb, Vasilis Friderikos, and A Hamid Aghvami. Delay sensitive virtual network function placement and routing. In *2018 25th international conference on telecommunications (ICT)*, pages 394–398. IEEE, 2018.

[44] Jalal Khamse-Ashari, Gamini Senarath, Irem Bor-Yaliniz, and Halim Yanikomeroglu. An agile and distributed mechanism for inter-domain network slicing in next-generation mobile networks. *IEEE Transactions on Mobile Computing*, 2021.

[45] Francesca Fossati, Stefano Moretti, Patrice Perny, and Stefano Secci. Multi-resource allocation for network slicing. *IEEE/ACM Transactions on Networking*, 28(3):1311–1324, 2020.

[46] Hassan Halabian. Distributed resource allocation optimization in 5g virtualized networks. *IEEE Journal on Selected Areas in Communications*, 37(3):627–642, 2019.

[47] Mohammad M Tajiki, Stefano Salsano, Luca Chiaraviglio, Mohammad Shojafar, and Behzad Akbari. Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining. *IEEE Transactions on Network and Service Management*, 16(1):374–388, 2018.

[48] Urmila M Diwekar. *Introduction to applied optimization*, volume 22. Springer Nature, 2020.

[49] Martin Aruldoss, T Miranda Lakshmi, and V Prasanna Venkatesan. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1):31–43, 2013.

[50] Gong Mao-Guo, Jiao Li-Cheng, Yang Dong-Dong, and Ma Wen-Ping. Evolutionary multi-objective optimization algorithms. 2009.

[51] Mark Voorneveld. Characterization of pareto dominance. *Operations Research Letters*, 31(1):7–11, 2003.

[52] Antonio López Jaimes, Saúl Zapotecas Martınez, Carlos A Coello Coello, et al. An introduction to multiobjective optimization techniques. *Optimization in Polymer Processing*, pages 29–57, 2009.

[53] Carlos A Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems*, 1(3):269–308, 1999.

[54] H Monsef, M Naghashzadegan, Ali Jamali, and Raziyeh Farmani. Comparison of evolutionary multi objective optimization algorithms in optimum design of water distribution network. *Ain Shams Engineering Journal*, 10(1):103–111, 2019.

[55] CA Coello Coello and Maximino Salazar Lechuga. Mopso: A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1051–1056. IEEE, 2002.

[56] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[57] Bram L Gorissen, İhsan Yanıkoğlu, and Dick den Hertog. A practical guide to robust optimization. *Omega*, 53:124–137, 2015.

[58] Zukui Li and Christodoulos A Floudas. Robust counterpart optimization: Uncertainty sets, formulations and probabilistic guarantees. In *proceedings of the 6th conference on foundations of computer-aided process operations, Savannah (Georgia)*, 2012.

[59] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[60] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[61] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[62] Qiang Liu and Tao Han. When network slicing meets deep reinforcement learning. In *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*, pages 29–30, 2019.

[63] Haozhe Wang, Yulei Wu, Geyong Min, Jie Xu, and Pengcheng Tang. Data-driven dynamic resource scheduling for network slicing: A deep reinforcement learning approach. *Information Sciences*, 498:106–116, 2019.

[64] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Deep reinforcement learning: an overview. In *Proceedings of SAI Intelligent Systems Conference*, pages 426–440. Springer, 2016.

[65] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[66] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[67] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[68] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[69] Zidong Zhang, Dongxia Zhang, and Robert C Qiu. Deep reinforcement learning for power system applications: An overview. *CSEE Journal of Power and Energy Systems*, 6(1):213–225, 2019.

[70] Forest Agostinelli, Guillaume Hocquet, Sameer Singh, and Pierre Baldi. From reinforcement learning to deep reinforcement learning: An overview. *Braverman readings in machine learning. key ideas from inception to current state*, pages 298–328, 2018.

[71] Wuhui Chen, Xiaoyu Qiu, Ting Cai, Hong-Ning Dai, Zibin Zheng, and Yan Zhang. Deep reinforcement learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2021.

[72] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[73] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[74] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[75] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[76] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[78] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[79] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[80] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(4):3133–3174, 2019.

[81] Chen Qi, Yuxiu Hua, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang. Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing. *IEEE Communications Letters*, 23(8):1337–1341, 2019.

[82] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[83] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[84] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[85] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[86] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asyn-

chronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[87] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[88] Yizhen Xu, Zhengyang Zhao, Peng Cheng, Zhuo Chen, Ming Ding, Branka Vucetic, and Yonghui Li. Constrained reinforcement learning for resource allocation in network slicing. *IEEE Communications Letters*, 25(5):1554–1558, 2021.

[89] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[90] Rongpeng Li, Chujie Wang, Zhifeng Zhao, Rongbin Guo, and Honggang Zhang. The lstm-based advantage actor-critic learning for resource management in network slicing with user mobility. *IEEE Communications Letters*, 24(9):2005–2009, 2020.

[91] Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. *arXiv preprint arXiv:1706.06084*, 2017.

[92] David G Andersen. Theoretical approaches to node assignment. 2002.

[93] Tarik Taleb, Badr Mada, Marius-Iulian Corici, Akihiro Nakao, and Hannu Flinck. Permit: Network slicing for personalized 5g mobile telecommunications. *IEEE Communications Magazine*, 55(5):88–93, 2017.

[94] Xuan Zhou, Rongpeng Li, Tao Chen, and Honggang Zhang. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine*, 54(7):146–153, 2016.

[95] Wanqing Guan, Xiangming Wen, Luhan Wang, Zhaoming Lu, and Yidi Shen. A service-oriented deployment policy of end-to-end network slicing based on complex network theory. *IEEE access*, 6:19691–19701, 2018.

[96] Gang Wang, Gang Feng, Tony QS Quek, Shuang Qin, Ruihan Wen, and Wei Tan. Reconfiguration in network slicing—optimizing the profit and performance. *IEEE Transactions on Network and Service Management*, 16(2):591–605, 2019.

[97] EL Hocine Bouzidi, Abdelkader Outtagarts, Abdelkrim Hebbar, Rami Langar, and Raouf Boutaba. Online based learning for predictive end-to-end network slicing in 5g networks. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.

[98] Wesley da Silva Coelho, Amal Benhamiche, Nancy Perrot, and Stefano Secci. On the impact of novel function mappings, sharing policies, and split settings in network slice design. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[99] Shan Yin, Zhan Zhang, Chen Yang, Yaqin Chu, and Shanguo Huang. Prediction-based end-to-end dynamic network slicing in hybrid elastic fiber-wireless networks. *Journal of Lightwave Technology*, 39(7):1889–1899, 2020.

[100] Uta Priss. Formal concept analysis in information science. *Annu. Rev. Inf. Sci. Technol.*, 40(1):521–543, 2006.

[101] Russ C Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, volume 1, pages 84–88. IEEE, 2000.

[102] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1):49–71, 2003.

[103] Stefano Coniglio, Arie Koster, and Martin Tieves. Data uncertainty in virtual network embedding: robust optimization and protection levels. *Journal of Network and Systems Management*, 24(3):681–710, 2016.

[104] Andreas Baumgartner, Thomas Bauschert, Fabio D'Andreagiovanni, and Varun S Reddy. Towards robust network slice design under correlated demand uncertainties. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[105] Ruihan Wen, Gang Feng, Jianhua Tang, Tony QS Quek, Gang Wang, Wei Tan, and Shuang Qin. On robustness of network slicing for next-generation mobile networks. *IEEE Transactions on Communications*, 67(1):430–444, 2018.

[106] Thomas Bauschert, Christina Büsing, Fabio D'Andreagiovanni, Arie MCA Koster, Manuel Kutschka, and Uwe Steglich. Network planning under demand uncertainty with robust optimization. *IEEE Communications Magazine*, 52(2):178–185, 2014.

[107] Peng Yang, Xing Xi, Kun Guo, Tony QS Quek, Jingxuan Chen, and Xianbin Cao. Proactive uav network slicing for urllc and mobile broadband service multiplexing. *IEEE Journal on Selected Areas in Communications*, 39(10):3225–3244, 2021.

[108] Haider D Resin Albonda and Jordi Pérez-Romero. An efficient ran slicing strategy for a heterogeneous network with embb and v2x services. *IEEE access*, 7:44771–44782, 2019.

[109] Dimitris Bertsimas and David B Brown. Constructing uncertainty sets for robust linear optimization. *Operations research*, 57(6):1483–1495, 2009.

[110] Sunday O Oladejo and Olabisi E Falowo. Profit-aware resource allocation for 5g sliced networks. In *2018 European Conference on Networks and Communications (EuCNC)*, pages 43–9. IEEE, 2018.

[111] Mojtaba Ahmadieh Khanesar, Mohammad Teshnehlab, and Mahdi Aliyari Shoorehdeli. A novel binary particle swarm optimization. In *2007 Mediterranean conference on control & automation*, pages 1–6. IEEE, 2007.

[112] Joel Goh and Melvyn Sim. Robust optimization made easy with rome. *Operations Research*, 59(4):973–985, 2011.

[113] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[114] Sen Su, Zhongbao Zhang, Alex X Liu, Xiang Cheng, Yiwen Wang, and Xinchao Zhao. Energy-aware virtual network embedding. *IEEE/ACM Transactions on Networking*, 22(5):1607–1620, 2014.

[115] Rongping Lin, Liu He, Shan Luo, and Moshe Zukerman. Energy-aware service function chaining embedding in nfv networks. *IEEE Transactions on Services Computing*, 2022.

[116] Anouar Rkhami, Tran Anh Quang Pham, Yassine Hadjadj-Aoul, Abdelkader Outtagarts, and Gerardo Rubino. On the use of graph neural networks for virtual network embedding. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2020.

[117] Ye Tian, Xingyi Zhang, Chao Wang, and Yaochu Jin. An evolutionary algorithm for large-scale sparse multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(2):380–393, 2019.

[118] Ramon A Gallego, Rubén Romero, and Alcir J Monticelli. Tabu search algorithm for network synthesis. *IEEE Transactions on Power Systems*, 15(2):490–495, 2000.

[119] Kyungjoo Suh, Sunwoo Kim, Yongjun Ahn, Seungnyun Kim, Hyungyu Ju, and Byonghyo Shim. Deep reinforcement learning-based network slicing for beyond 5g. *IEEE Access*, 2022.

[120] Taihui Li, Xiaorong Zhu, and Xu Liu. An end-to-end network slicing algorithm based on deep q-learning for 5g network. *IEEE Access*, 8:122229–122240, 2020.

[121] Qiang Liu, Tao Han, Ning Zhang, and Ye Wang. Deepslicing: Deep reinforcement learning assisted resource allocation for network slicing. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[122] Ying Wang, Naling Li, Peng Yu, Wenjing Li, Xuesong Qiu, Shangguang Wang, and Mohamed Cheriet. Intelligent and collaborative orchestration of network slices. *IEEE Transactions on Services Computing*, 2022.

[123] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[124] Rongpeng Li, Zhifeng Zhao, Qi Sun, I Chih-Lin, Chenyang Yang, Xianfu Chen, Minjian Zhao, and Honggang Zhang. Deep reinforcement learning for resource management in network slicing. *IEEE Access*, 6:74429–74441, 2018.

[125] Fengsheng Wei, Gang Feng, Yao Sun, Yatong Wang, Shuang Qin, and Ying-Chang Liang. Network slice reconfiguration by exploiting deep reinforcement learning with large action space. *IEEE Transactions on Network and Service Management*, 17(4):2197–2211, 2020.

[126] Almuthanna Nassar and Yasin Yilmaz. Deep reinforcement learning for adaptive network slicing in 5g for intelligent vehicular systems and smart cities. *IEEE Internet of Things Journal*, 9(1):222–235, 2021.

[127] Yaser Azimi, Saleh Yousefi, Hashem Kalbkhani, and Thomas Kunz. Energy-efficient deep reinforcement learning assisted resource allocation for 5g-ran slicing. *IEEE Transactions on Vehicular Technology*, 2021.

[128] Yohan Kim and Hyuk Lim. Multi-agent reinforcement learning-based resource management for end-to-end network slicing. *IEEE Access*, 9:56178–56190, 2021.

[129] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[130] Sen Wang, Jun Bi, Jianping Wu, Athanasios V Vasilakos, and Qilin Fan. Vne-td: A virtual network embedding algorithm based on temporal-difference learning. *Computer Networks*, 161:251–263, 2019.

[131] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[132] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[133] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[134] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[135] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[136] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[137] Ying Yuan, Zejie Tian, Cong Wang, Fanghui Zheng, and Yanxia Lv. A q-learning-based approach for virtual network embedding in data center. *Neural Computing and Applications*, 32(7):1995–2004, 2020.

[138] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[139] Ruxandra F Olimid and Gianfranco Nencioni. 5g network slicing: A security overview. *IEEE Access*, 8:99999–100009, 2020.