

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Network Slice Embedding with Flexible Configurations in 5G Networks and Beyond

Quang-Trung Luu, Do-Minh Tran, Minh-Thanh Nguyen, Michel Kieffer,
Dinh Thai Hoang, Tai-Hung Nguyen, Huu-Thanh Nguyen, and Van-Dinh Nguyen

Abstract—Network slicing enables the creation of multiple virtual networks (*i.e.*, slices) over a shared network infrastructure, each tailored to a specific service. A key challenge lies in network slice embedding, which maps virtual network functions (VNFs) and links onto the physical network. Unlike prior works that assumed fixed configurations, we design a flexible system that allows for selecting the best configuration for each slice based on current physical resource availability during embedding. This leads to a joint optimization of (*i*) slice configuration selection (SCS) and (*ii*) slice admission control and embedding (SACE). To solve this, we propose two approaches: an exact method that formulates the joint SCS-SACE problem as an integer linear program (ILP), and a scalable alternative that decouples the problem, solving SCS via reinforcement learning and SACE via either ILP or a heuristic. Simulation results show that allowing flexible configuration selection improves slice acceptance by up to 10%, enabling more efficient slice deployment in resource-constrained networks.

Index Terms—Network slicing, flexible configuration, slice embedding, 5G and beyond, reinforcement learning.

I. INTRODUCTION

In 5G and beyond, network slicing emerged as an enabling technology for the creation of multiple virtual networks over a common physical infrastructure [1]. The demands of various applications and services, ranging from high-bandwidth video streaming to low-latency industrial control systems, can be efficiently satisfied by customized virtual networks, or *slices*, built on top of a shared physical network infrastructure. This customization is accomplished by dedicating specific resources of the physical network to virtual network functions (VNFs) and configuring them to provide a service satisfying some desired metrics, *e.g.*, bandwidth, latency, and reliability [2].

One of the critical challenges in network slicing is the problem of network slice embedding, also called virtual network embedding or service function chain (SFC) embedding. This task involves optimally mapping VNFs and their interconnections, known as *virtual links*, onto the physical network

Q.-T. Luu, D.-M. Tran, M.-T. Nguyen, T.-H. Nguyen, and H.-T. Nguyen are with School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, Hanoi 100000, Vietnam. Q.-T. Luu is also with Université Paris-Saclay - CNRS - CentraleSupélec - L2S, Gif-sur-Yvette, F-91192, France. Corresponding author: Quang-Trung Luu (e-mail: quangtrung.luu@centralesupelec.fr).

M. Kieffer is with Université Paris-Saclay - CNRS - CentraleSupélec - L2S, Gif-sur-Yvette, F-91192, France (e-mail: michel.kieffer@centralesupelec.fr).

D. T. Hoang is with School of Electrical and Data Engineering, University of Technology Sydney, Australia (e-mail: hoang.dinh@uts.edu.au).

V.-D. Nguyen is with the School of Computer Science and Statistics, Trinity College Dublin, The University of Dublin, Dublin 2, D02 PN40, Ireland (e-mail: dinh.nguyen@tcd.ie).

This research is funded by Hanoi University of Science and Technology (HUST) under project number T2023-TD-011.

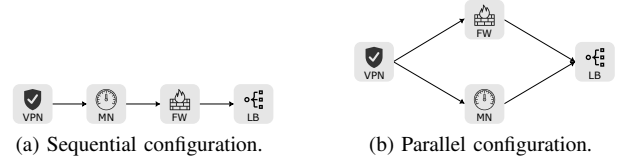


Fig. 1. An example of network slicing with two possible configurations: (a) sequential and (b) parallel configuration, both are dedicated to security and performance optimization for network traffic in data centers [3], [4].

infrastructure. VNFs are software-based network functions that replace traditional hardware-based equipments, providing enhanced flexibility and scalability. Embedding these VNFs onto the physical network necessitates careful consideration of various factors, such as resource availability, network topology, and satisfaction of service level agreements (SLAs). An efficient embedding has to ensure an efficient utilization of network resources, meet the performance requirements of each slice, and minimize operational costs. Nevertheless, finding the optimal embedding remains computationally challenging, especially in large-scale and dynamic networks [1], [2].

Most prior works on network slicing assume that the slice has a fixed configuration, meaning its VNFs are organized in a predetermined way [5], [6]. In practice, certain VNFs, for example, the distributed unit (DU) in an open radio access network (Open RAN), must be anchored at the beginning of the slice to guarantee low-latency and highly reliable connections [7]. In contrast, other VNFs may be placed with greater flexibility, resulting in multiple possible slice configurations (see Fig. 1). These configurations can involve different orders of VNFs, various types of VNFs, or even diverse structures for composing the VNFs within a slice, such as linear, branched, or cyclic forms [8], [9]. This flexibility provides operators with additional degrees of freedom to adjust virtual network configurations in response to changing traffic patterns and service demands, to optimize resource utilization, and to improve overall network performance. It also provides more flexibility to solve the network slice embedding problem.

The key contributions of this letter are as follows.

- We first introduce a novel optimization framework for network slice embedding (NSE) with flexible configuration selection. Unlike prior works assuming fixed configurations, our approach enables the system to select the best configuration for each slice based on the current physical resource availability during slice embedding. This leads to a new joint optimization of (*i*) slice configuration selection (SCS) and (*ii*) slice admission control and embedding (SACE), which is an *NP*-hard nonlinear integer programming (INLP) problem;

- To address this problem effectively, we propose two solution approaches. The first approach computes optimal solutions by formulating the joint SCS and SACE problem as an integer linear program (ILP). This is achieved by applying a series of linearization techniques to transform the original nonlinear integer programming (INLP) formulation into an ILP. The second approach targets scalability and near-optimality in large-scale scenarios by decoupling the problem into two subproblems: SCS is handled by reinforcement learning (RL), while SACE is solved via either an ILP solver or a heuristic;
- We perform comprehensive simulations with different slice settings and various network topologies to evaluate the performance of our proposed approaches. The results demonstrate that, allowing flexible slice configurations significantly improves slice acceptance rates with a performance gap of up to 10%. This leads to enhanced resource utilization in constrained network environments.

II. PROBLEM FORMULATION

A. Network and Slice Model

The physical network can be represented by a weighted graph denoted by $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of physical nodes and $\mathcal{L}$ is the set of physical links. Each physical node $i \in \mathcal{N}$ is characterized by its resource capacity A_i (e.g., computing, storage, or processing). Similarly, each physical link $ij \in \mathcal{L}$ has a resource capacity of A_{ij} (e.g., bandwidth).

For each network slice $s \in \mathcal{S}$, we denote $\mathcal{K}_s$ as the set of all configurations that the slice can take during the embedding process. Each configuration $k \in \mathcal{K}_s$ is represented by a weighted graph $\mathcal{G}_s^k = (\mathcal{N}_s^k, \mathcal{L}_s^k)$, where $\mathcal{N}_s^k$ and $\mathcal{L}_s^k$ respectively represent the set of VNFs and virtual links in configuration k of slice s . Each VNF $v \in \mathcal{N}_s^k$ requires an amount R_v of node resources. Similarly, each virtual link $vw \in \mathcal{L}_s^k$ is characterized by a resource demand R_{vw} .

B. Variables

The problem of resource allocation for network slicing with flexible configuration involves the joint optimization of slice configuration selection (SCS) and slice admission control and embedding (SACE), as illustrated in Fig. 2. To formulate this problem, we introduce three sets of binary variables

$$\boldsymbol{\pi} = \{\pi_s\}_{s \in \mathcal{S}}; \quad \boldsymbol{\Phi} = \{\Phi_s^k\}_{k \in \mathcal{K}_s, s \in \mathcal{S}}; \quad \text{and} \quad (1a)$$

$$\boldsymbol{\theta} = \{\theta_i^{v,k,s}, \theta_{ij}^{vw,k,s}\}_{(i,ij) \in \mathcal{G}, (v,vw) \in \mathcal{G}_s^k, k \in \mathcal{K}_s, s \in \mathcal{S}}. \quad (1b)$$

Here, π_s is the slice admission control variable, indicating whether slice s is accepted. The configuration selection variable Φ_s^k indicates whether configuration k is selected to deploy slice s . Finally, $\theta_i^{v,k,s}$ and $\theta_{ij}^{vw,k,s}$ indicate whether the virtual node v and virtual link vw of configuration k are mapped onto physical node i and physical link ij , respectively.

C. Constraints and Formulation

A valid solution to the joint SCS and SACE problem has to satisfy the following constraints.

Only one configuration. To enforce that a single configuration k is selected for any accepted slice s , one has

$$\sum_{k \in \mathcal{K}_s} \Phi_s^k = \pi_s, \quad \forall s \in \mathcal{S}. \quad (2)$$

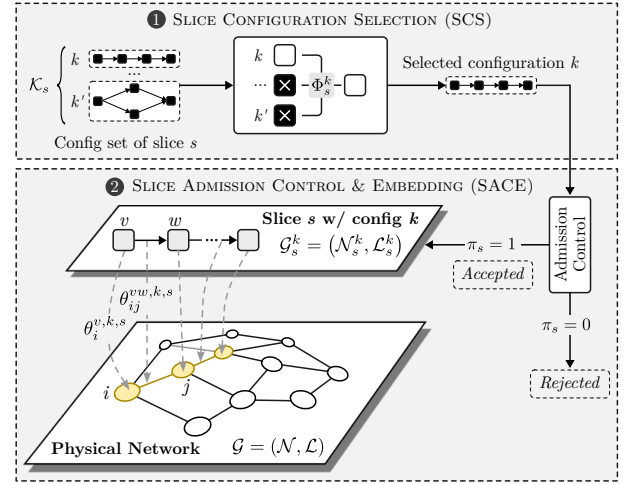


Fig. 2. Illustration of the joint optimization problem of (1) slice configuration selection (SCS) and (2) slice admission control and embedding (SACE), considering one single slice s .

Node capacity. The total resources that a given physical node $i \in \mathcal{N}$ allocates to VNFs of all configurations of all slices must not exceed its capacity A_i :

$$\sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}_s} \sum_{v \in \mathcal{N}_s^k} \theta_i^{v,k,s} R_v \leq A_i, \quad \forall i \in \mathcal{N}. \quad (3)$$

Link capacity. Similar to (3), the following constraint avoids the violation of link capacity limit,

$$\sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}_s} \sum_{vw \in \mathcal{L}_s^k} \theta_{ij}^{vw,k,s} R_{vw} \leq A_{ij}, \quad \forall ij \in \mathcal{L}. \quad (4)$$

Admitted slices are served. Once a slice s is accepted and a configuration k has been selected (i.e., $\pi_s = \Phi_s^k = 1$) to deploy s , all the VNFs of k must be mapped onto the physical network to deliver the dedicated service:

$$\sum_{i \in \mathcal{N}} \theta_i^{v,k,s} = \pi_s \Phi_s^k, \quad \forall v \in \mathcal{N}_s^k, k \in \mathcal{K}_s, s \in \mathcal{S}. \quad (5)$$

To linearize the quadratic term $\pi_s \Phi_s^k$, we introduce an additional variable $z_s^k \in \{0, 1\}$ and the following constraints:

$$z_s^k \leq \pi_s; \quad z_s^k \leq \Phi_s^k; \quad z_s^k \geq \pi_s + \Phi_s^k - 1, \quad \forall k \in \mathcal{K}_s, s \in \mathcal{S} \quad (6)$$

to ensure that $z_s^k = 1$ if and only if both π_s and Φ_s^k are equal to 1. Having this, (5) now becomes

$$\sum_{i \in \mathcal{N}} \theta_i^{v,k,s} = z_s^k, \quad \forall v \in \mathcal{N}_s^k, k \in \mathcal{K}_s, s \in \mathcal{S}. \quad (7)$$

It is worth noting that (7) also guarantees mapping uniqueness: when $z_s^k = 1$, the binary nature of $\theta_i^{v,k,s}$ ensures that each VNF is mapped onto exactly one physical node.

Flow conservation. The following constraint guarantees the conservation of slice's traffic,

$$\sum_{j \in \mathcal{N}} (\theta_{ij}^{vw,k,s} - \theta_{ji}^{vw,k,s}) - (\theta_i^{v,k,s} - \theta_i^{w,k,s}) = 0, \quad \forall i \in \mathcal{N}, vw \in \mathcal{L}_s^k, s \in \mathcal{S}, k \in \mathcal{K}_s : \Phi_s^k = 1. \quad (8)$$

Precisely, (8) ensures continuous connection between each pair of virtual nodes in configuration k of slice s on the

physical network. As indicated in [10], slices of any topology (sequential, parallel, or branched) can be handled by (8). It can be seen that (8) is a conditional constraint (only activated when $\Phi_s^k = 1$). To address this, we use the big- M method as:

$$-M(1 - \Phi_s^k) \leq \text{LHS}_{(8)} \leq M(1 - \Phi_s^k), \quad \forall i \in \mathcal{N}, vw \in \mathcal{L}_s^k, k \in \mathcal{K}_s, s \in \mathcal{S} \quad (9)$$

where $\text{LHS}_{(8)}$ denotes the left-hand side of (8). Here, M is chosen as an upper bound on the maximum possible value of $\text{LHS}_{(8)}$. Since all involved variables are binary, this term is bounded by the node degree, and in a network with $|\mathcal{N}|$ nodes we set $M = |\mathcal{N}|$.

Final formulation. One denotes by $N(\pi)$ the slice acceptance rate and by $L(\theta)$ the average number of used physical links per slice. Expressions of $N(\pi)$ and $L(\theta)$ are given by

$$N(\pi) = \sum_{s \in \mathcal{S}} \frac{\pi_s}{|\mathcal{S}|}; L(\theta) = \sum_{s \in \mathcal{S}} \sum_{k \in \mathcal{K}_s} \sum_{vw \in \mathcal{L}_s^k} \sum_{ij \in \mathcal{L}} \frac{\theta_{ij}^{vw,k,s}}{|\mathcal{L}||\mathcal{S}|}. \quad (10)$$

Finally, the joint SCS and SACE optimization problem, which aims to maximize the slice acceptance rate and minimize the link usage, accounting for flexible slice configuration, can be formulated as

$$\max_{\pi, \theta, \Phi} f(\pi, \theta) \quad \text{s.t. (2), (3), (4), (6), (7), (9)}. \quad (\mathcal{P}_1)$$

where $f(\pi, \theta) = \alpha N(\pi) - (1 - \alpha) L(\theta)$ and $\alpha \in [0, 1]$ is a tuning parameter to adjust the importance of $N(\pi)$ and $L(\theta)$. Similar to the NSE problem with fixed configuration, $(\mathcal{P}_1)$ can also be proven to be NP-hard [11]. Consequently, solving directly the ILP in $(\mathcal{P}_1)$ is difficult and does not scale, especially when dealing with large $|\mathcal{G}|$ and $|\mathcal{S}|$. In Sec. III, we introduce our proposed approach to obtain near-optimal solutions to $(\mathcal{P}_1)$.

III. PROPOSED SOLUTIONS

We propose two approaches to solve $(\mathcal{P}_1)$: exact and near-optimal. The optimal approach involves solving $(\mathcal{P}_1)$ directly using an ILP solver, serving as a performance benchmark to evaluate the solution quality. Alternatively, the near-optimal approach is designed for practical deployment in large-scale scenarios. It decomposes $(\mathcal{P}_1)$ into two subproblems, SCS and SACE, and applies a tailored solution for each as follows. *First*, for the SCS subproblem, an RL framework based on Q-learning (QL) is designed, leveraging its capability to handle dynamic decision-making. *Second*, the SACE subproblem can be solved either by using an ILP solver for optimal solutions or by applying a heuristic to ensure computational efficiency. This decomposition enables a near-optimal approach that balances solution quality and computational complexity.

A. Solving the SCS Subproblem

We employ Q-learning, a widely used model-free RL method [12], [13], to optimize the SCS process by developing an optimal policy for selecting slice configurations based on their performance in slice acceptance rate and physical resource usage. Specifically, this algorithm tries to embed each slice s with configuration k sequentially onto the physical network $\mathcal{G}$ and selects the best configuration for each slice so that the

later SACE process is facilitated. The RL model for the QL algorithm is introduced in what follows.

Agent. We consider a centralized approach, where the RL agent resides in a centralized service management and orchestration (SMO) entity. For instance, in the O-RAN architecture, the RL agent could be part of the radio access network controller [14]. This location provides the RL agent the ability to access real-time data from the physical network.

State. Let $\mathcal{S} = \{\mathbf{s}_t \in \mathcal{S}\}_{\forall t}$ be the set of possible states of the environment. At decision step t , the state $\mathbf{s}_t$ is given by $\mathbf{s}_t = \mathcal{S}_t$ where $\mathcal{S}_t$ denotes the slice of index t in the slice set $\mathcal{S}$ that is currently being processed.

Action. Based on the observation of current states, the RL agent takes an action $\mathbf{a}$ from the action set $\mathcal{A}$. This set includes all possible configurations k of slice s : $\mathcal{A} = \{\mathbf{a} \in \mathcal{K}_s; -1\}$, wherein $\mathbf{a} = -1$ indicates the *skip* action, i.e., the current processing is ignored due to infeasible embedding.

Reward. After selecting an action (i.e., slice configuration), slice embedding is carried out using the corresponding embedding algorithm. If the embedding is successful, the RL agent receives a reward $\mathbf{r}_t = f(\pi, \theta)$, which corresponds to the objective value of $(\mathcal{P}_1)$. This reward reflects the solution quality, where higher values indicate better embedding performance, i.e., higher slice acceptance rates and lower physical link utilization. If the embedding fails, the agent receives a penalty $\mathbf{r}_t = -0.5$. If the agent skips embedding the current slice, a negative reward of $\mathbf{r}_t = -0.3$ is assigned. This allows the RL agent to learn to skip difficult embeddings in subsequent training episodes, leading to faster convergence.

QL training. The training process of the Q-learning algorithm employs an ε -greedy strategy [13] to balance exploration and exploitation. Once an action has been taken, i.e., a configuration has been chosen, it is sent by the RL agent to the environment and the slice embedding process begins. Once the process is finished, the environment returns the next state $\mathbf{s}_t$ and the corresponding reward $\mathbf{r}$ to the RL agent. Next, to enhance the decision-making process over time, the RL agent updates the Q-table using

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1 - \lambda) Q(\mathbf{s}_t, \mathbf{a}_t) + \lambda (\mathbf{r}_t + \gamma \max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a})) \quad (11)$$

where $Q(\mathbf{s}_{t+1}, \mathbf{a})$ is the Q-value obtained from taking action $\mathbf{a}$ in state $\mathbf{s}_t$; $\mathcal{R}_t$ is the reward received after taking action $\mathbf{a}$; $\max_{\mathbf{a} \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a})$ is the maximum Q-value for the next state $\mathbf{s}_{t+1}$; λ is the learning rate; and γ is the discount factor. By updating this Q-table iteratively, the RL agent can improve its Q-values, enabling it to make better decisions in future interactions with the environment.

B. Solving the SACE Subproblem

As discussed in Sec. III, the SACE subproblem can be solved either by using an ILP solver for exact solutions or by a scalable algorithm. For the latter, we employ a greedy algorithm, namely CLC (slice embedding using Configuration with Least Cost) that helps find near-optimal solutions to $(\mathcal{P}_1)$. CLC tries to embed each configuration onto the physical network in a greedy manner, i.e., for each VNF, select the physical node that has the largest capacity and has not yet been used for the current slice. The embedding of virtual links between each

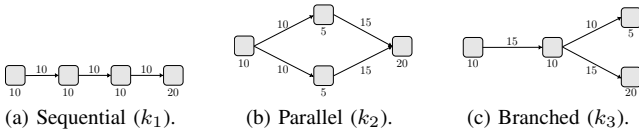


Fig. 3. Three slice configurations used for the simulations: (a) sequential (k_1), (b) parallel (k_2), and (c) branched (k_3) configuration, inspired by [3]. The number at each node indicates the required number of CPUs, the number at each link indicates the required bandwidth, measured in Mbps.

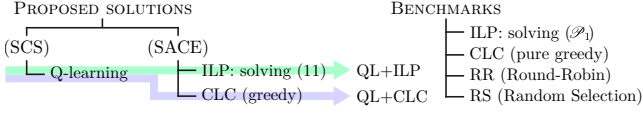


Fig. 4. Summary of the compared algorithms.

pair of VNFs is then performed using the Dijkstra algorithm. After trying to map all of the available configurations in $\mathcal{K}_s$ of slice s , CLC accepts s ($\pi_s = 1$) and selects the configuration k yielding the best objective value to $(\mathcal{P}_1)$ to deploy s . If there exists no mappable configuration, CLC rejects slice s ($\pi_s = 0$) and continues with the next slice in $\mathcal{S}$.

C. Complexity Analysis

ILP requires solving $(\mathcal{P}_1)$ over the entire slice set $\mathcal{S}$, resulting in an optimization problem with $|\mathcal{S}| + \sum_s |\mathcal{K}_s| + \sum_{s,k} (|\mathcal{N}| |\mathcal{N}_s^k| + |\mathcal{L}| |\mathcal{L}_s^k|)$ variables and $|\mathcal{N}| + |\mathcal{L}| + |\mathcal{S}| + 3 \sum_s |\mathcal{K}_s| + \sum_{s,k} (|\mathcal{N}_s^k| + 2 |\mathcal{N}| |\mathcal{L}_s^k|)$ constraints. In contrast, QL+ILP decomposes the problem into $|\mathcal{S}|$ subproblems, each involving $1 + |\mathcal{K}_s| + \sum_k (|\mathcal{N}| |\mathcal{N}_s^k| + |\mathcal{L}| |\mathcal{L}_s^k|)$ variables and $1 + |\mathcal{N}| + |\mathcal{L}| + 3 |\mathcal{K}_s| + \sum_k (|\mathcal{N}_s^k| + 2 |\mathcal{N}| |\mathcal{L}_s^k|)$ constraints.

On the other hand, the RL component of QL+ILP and QL+CLC follows a standard Q-learning algorithm, whose convergence under bounded rewards and sufficient exploration has been well established in the literature [12], [13].

IV. PERFORMANCE EVALUATION

A. System Setup

Physical network. Three network topologies from the SNDlib library [15] are considered for our simulations: ABILENE, COST266, and TA2. The number of physical nodes and links $(|\mathcal{N}|, |\mathcal{L}|)$ of these networks are resp. (12, 30), (37, 114), and (65, 216). For the first evaluation (Sec. IV-B), we consider an infrastructure network where each physical node has 20 CPUs and each link has 20 Mbps of available bandwidth. For the second evaluation (Sec. IV-C), the available resources at nodes and links are randomly generated within the range [15, 25] and the reported results are averaged over 10 runs.

Slices. Inspired by the example in Fig. 1, we use three slice configurations for the simulations: sequential (k_1), parallel (k_2), and branched (k_3) topologies, as shown in Fig. 3.

Benchmarks. Alongside ILP and CLC, we also consider RR (round-robin) and RS (random selection) as baseline benchmarks for the evaluation. RR applies round-robin selection for slice configurations and physical nodes, while RS relies on random selection. Both use Dijkstra's algorithm for virtual link embedding and terminate when all slices are embedded or resources are exhausted. The compared algorithms are summarized in Fig. 4.

Simulation scenarios. Three scenarios are considered: small-

scale ($|\mathcal{S}| = 10$) on ABILENE; medium-scale ($|\mathcal{S}| = 30$) on COST266; and large-scale ($|\mathcal{S}| = 40$) on TA2. The small-scale scenario evaluates both fixed (k_1 -only, k_2 -only, k_3 -only) and flexible settings (any configuration). Medium- and large-scale scenarios consider only the flexible setting.

Metrics. Evaluation is based on three metrics: (i) slice acceptance rate $N(\pi)$, (ii) average physical link usage per slice $L(\theta)$, and (iii) computation time.

RL parameters. The RL agent is trained with an initial ε of 1.0, which decays by a factor of 1.98×10^{-5} per episode. The values of γ and λ are chosen through convergence analysis. Fig. 5a shows the convergence behavior of QL on the ABILENE network with varying γ and λ . The average reward converges after 100–150 episodes across all settings. Accordingly, we select $(\lambda, \gamma) = (0.099, 0.1)$, which achieves the highest average reward, to evaluate QL+ILP and QL+CLC.

Fig. 5b presents the convergence behavior across different network topologies with random resource availability. Despite differences in network size and topology, the QL agent consistently converges within at most 200 episodes, exhibiting its robustness and scalability.

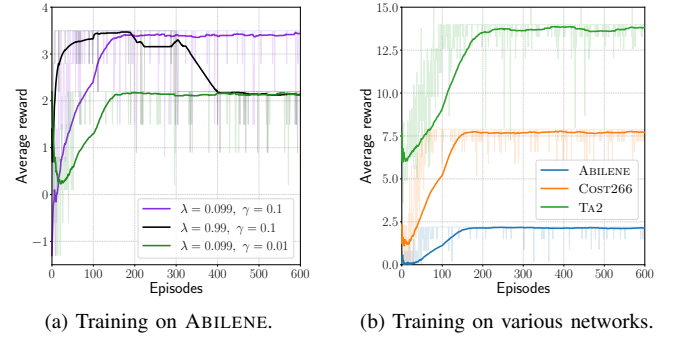


Fig. 5. Convergence behavior of QL using (a) the ABILENE network with fixed available resources and varying λ and γ ; and (b) various networks with randomly chosen resources.

Other parameters and setup. The tuning parameter α is set to 0.9999 in ILP. Dijkstra's algorithm is used in RR and RS with a maximum hop count of 4. Simulations are implemented in Python 3.10 on an Intel i5-12500T PC with 32 GB RAM. ILP-based methods involve Gurobi v11.0.3.

B. Impact of Flexible Slice Configuration

Fig. 6a shows the acceptance rate of ILP using the ABILENE network under different slice settings. It can be seen that, performing ILP with flexible setting consistently achieves better or equal performance across all topologies compared to the fixed configuration settings (k_1 -only and k_2 -only). The flexible setting leads to an improvement of 10% compared to the fixed settings (k_1 -only, k_2 -only, and k_3 -only). Fig. 6b further shows that the flexible setting also maintains a good balance between high acceptance rate and moderate link utilization, achieving lower link usage than the k_1 -only and k_3 -only settings, and comparable performance to the k_2 -only setting. Overall, the results confirm that, allowing the flexibility in selecting slice configuration enables better adaptation to network resource availability and topological constraints.

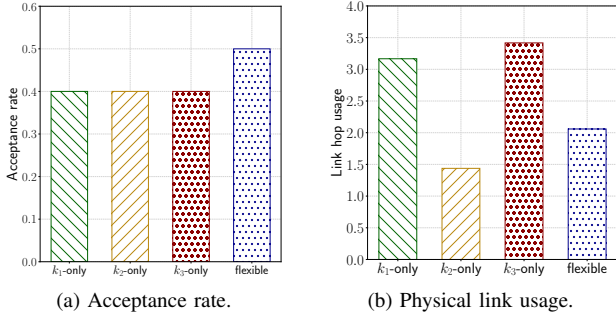


Fig. 6. (a) Acceptance rate and (b) physical link usage of ILP with different slice settings and network topologies.

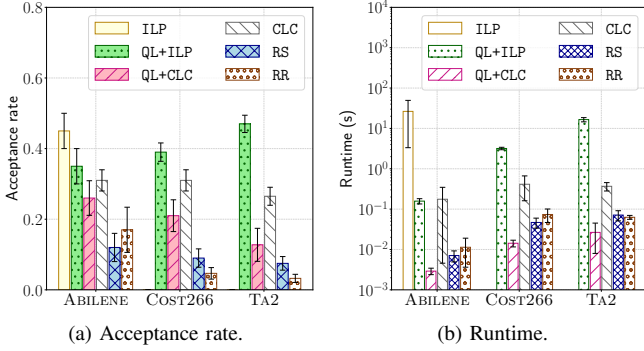


Fig. 7. (a) Acceptance rate and (b) runtime of the compared algorithms using different network topologies. Reported results are averaged over 10 runs. ILP is excluded from the evaluation on the TA2 and COST266 networks due to its computational complexity.

C. Performance Comparison in the Flexible Setting

Fig. 7 compares the acceptance rate and runtime of the evaluated algorithms under the flexible setting across different network topologies. On the ABILENE network (see Fig. 7a), QL+ILP achieves the closest acceptance rate to that of ILP (35% to 45%) while significantly reducing runtime, and clearly outperforms all other algorithms. On the larger TA2 and COST266 networks, ILP is excluded due to its high computational complexity, whereas the proposed QL-based methods remain scalable. In these larger topologies, the performance gain of QL+ILP becomes more pronounced, achieving the highest acceptance rates with a gap of up to 20.5% compared to the second-best method (CLC).

From Fig. 7b, it can be seen that ILP incurs significantly higher runtime even on the small network ABILENE, while QL+CLC achieves the lowest runtime among the learning-based methods and remains close to CLC, demonstrating an effective trade-off between performance and complexity.

Overall, QL+ILP offers a near-optimal trade-off between performance and complexity, while QL+CLC prioritizes speed, making these two suitable for different deployment scenarios.

V. CONCLUSIONS

In this letter, we introduced a novel optimization framework for network slice embedding with flexible configuration, jointly optimizing (i) slice configuration selection (SCS) and (ii) slice admission control and embedding (SACE). This leads to an NP-hard nonlinear integer program (INLP). To solve this, we propose two approaches. The first is an exact method that transforms the INLP into an integer linear program (ILP) using linearization techniques. The second is a scalable, near-optimal method that decouples the problem: SCS is solved

via Q-learning, while SACE is handled using either an ILP solver or a heuristic, resulting in two variants: QL+ILP and QL+CLC. Simulation results show that the exact ILP solution significantly improves slice acceptance and resource utilization, while the near-optimal methods offer a good balance between performance and computational efficiency, making them suitable for practical deployment.

REFERENCES

- [1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, 2017.
- [2] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang, and Z. Zhu, "Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models," *IEEE Netw.*, vol. 33, no. 6, pp. 172–179, 2019.
- [3] X. Lin, D. Guo, Y. Shen, G. Tang, and B. Ren, "DAG-SFC: Minimize the embedding cost of SFC with parallel VNFs," in *Proc. 47th International Conference on Parallel Processing*, 2018, pp. 1–10.
- [4] IETF, "Service Function Chaining Use Cases In Data Centers," *Internet-Draft draft-ietf-sfc-dc-use-cases-06*, 2017.
- [5] Q.-T. Luu, S. Kerboeuf, and M. Kieffer, "Admission control and resource reservation for prioritized slice requests with guaranteed SLA under uncertainties," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 3136–3153, 2022.
- [6] M.-T. Nguyen, Q.-T. Luu, T.-H. Nguyen, D.-M. Tran, T.-A. Do, K.-H. Do, and V.-H. Nguyen, "Accelerating network slice embedding with reinforcement learning," in *Proc. IEEE International Conference on Communications and Electronics (ICCE)*, 2024.
- [7] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Commun. Surv. Tutor.*, vol. 25, no. 2, pp. 1376–1411, 2023.
- [8] J. Huang, Q. Duan, S. Guo, Y. Yan, and S. Yu, "Converged network-cloud service composition with end-to-end performance guarantee," *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 545–557, 2015.
- [9] A. F. Ocampo, J. Gil-Herrera, P. H. Isolani, M. C. Neves, J. F. Botero, S. Latré, L. Zambenedetti, M. P. Barcellos, and L. P. Gaspary, "Optimal service function chain composition in network functions virtualization," in *Proc. 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security (AIMS)*, 2017, pp. 62–76.
- [10] R. Riggio, A. Bradai, D. Harutyunyan, T. Rasheed, and T. Ahmed, "Scheduling wireless virtual networks functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 2, pp. 240–252, 2016.
- [11] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
- [12] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.
- [13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [14] O-RAN Working Group 2, "AI/ML workflow description and requirements," *Tech. Rep. O-RAN.WG2.AIML-v01.03*, 2021.
- [15] S. Orłowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0-survivable network design library," *Networks*, vol. 55, no. 3, pp. 276–286, 2010.