Fragmentation-Aware VNF Placement: A Deep Reinforcement Learning Approach

Ramy Mohamed [*], Marios Avgeris [*], Aris Leivadeas [†], Ioannis Lambadaris [*]
[*] Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada
Email: ramy.mohamed@carleton.ca, mariosavgeris@cunet.carleton.ca, ioannis@sce.carleton.ca
[†] Department of Software and IT Engineering, École de technologie supérieure, Montreal, Canada
Email: aris.leivadeas@etsmtl.ca

***Abstract*—In this paper we address the challenge of efficiently deploying Virtual Network Functions (VNFs) in network infrastructures. This is particularly crucial when facing resource fragmentation, where available resources are not fully utilized due to the fluctuating allocation and deallocation of virtual network requests. Traditional optimization techniques often fall short in managing the dynamic complexities of VNF placement. To overcome this, we introduce a novel online VNF placement strategy using Deep Reinforcement Learning (DRL) combined with a Reward Constrained Policy Optimization (RCPO). This method leverages the flexibility of DRL and the constraint integration capacity of RCPO, ensuring compliance with performance and resource limitations while minimizing resource fragmentation. The results demonstrate that our DRL-based method surpasses existing methods, resulting in more effective resource management and less resource fragmentation.**

***Index Terms*—Resource Fragmentation, Neural Combinatorial Optimization, Reinforcement Learning, 5G, NFV, SFC, VNF Placement**

I. Introduction

Network Function Virtualization (NFV) transforms network functions into software instances on virtualized infrastructures, thus detaching them from specialized hardware. This concept visualizes network services as Service Function Chains (SFCs), comprising interconnected Virtual Network Functions (VNFs) [1]. The VNFs are vertices in the SFC graph, while the edges represent virtual links between them. Similarly, the underlying Network Function Virtualization Infrastructure (NFVI) is modeled as a graph with computing and forwarding nodes as vertices and physical communication links as edges. This creates the VNF Chain Placement Problem (VNF-CPP), i.e., mapping SFCs onto the physical network while optimizing costs and adhering to constraints [1] (Fig. 1).

VNF placement is inherently a complex and dynamic task, challenged by fluctuating network conditions and service demands, which deems traditional optimization methods like Integer Linear Programming (ILP) unsuitable [2], [3]. Thus, recent research has turned to Machine Learning -based techniques such as Deep Reinforcement Learning (DRL); for instance, in [4], authors focused on optimizing the acceptance ratio and minimizing the Age of Information (AoI) using DRL [4]. Moreover, in [5], authors have presented a DRL-based method for optimizing the power consumption of the VNF Placement by extending the Neural Combinatorial Optimization (NCO) theory and combining it with heuristics. NCO is

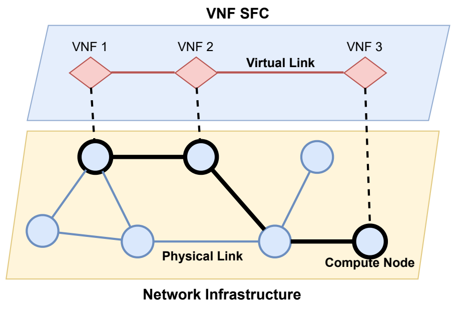

Fig. 1: The VNF Chain Placement Problem (VNF-CPP).

a machine learning approach that employs neural networks to solve combinatorial optimization problems by learning a representation of them. It is then used to make decisions or predictions about the optimal solution [6].

In this study, we investigate the problem of resource fragmentation, where the continuum of available infrastructure resources is used inefficiently because of unstrategic SFC placements, thus reducing capacity or performance. We introduce a novel online method for fragmentation-aware VNF placement, utilizing a combination of DRL and Reward Constrained Policy Optimization (RCPO) [7]. This technique combines the strengths of DRL in forming optimal policies within dynamic settings with the capabilities of RCPO in integrating constraints potentially set by network operators. The contribution of this paper is threefold: i) we develop a novel approach for fragmentation-aware VNF placement problem using DRL and RCPO; ii) we solve the fragmentation problem by applying a novel DRL approach enhanced with a RCPO, and iii) we provide insights into its scalability and adaptability for real-world scenarios.

The rest of the paper is organized as follows: Section II introduces Resource Fragmentation. Section III presents the problem formulation. Section IV explains how the problem is solved using DRL and RCPO. The experimental results are presented in Section V. Finally, Section VI concludes the paper.

II. Resource Fragmentation

The authors in [8] introduced the Resource Fragmentation Degree (RFD) metric. This metric is designed to quantitatively measure the status of resource fragmentation at substrate nodes and links. RFD assesses how scattered or isolated resources are in relation to their neighboring entities, either nodes or links. To determine the RFD, the authors introduced two primary concepts: i) the connectivity of substrate nodes and ii) the connectivity of substrate links.

The connectivity of substrate nodes measures how connected a node is based on the distance (in hops) to other nodes, the residual resources (e.g., CPU capacity) available at each of these nodes, and the bandwidth resources available on the paths to these nodes. The following equation computes connectivity κ_j^n for any given node n_j in the network:

$$\kappa_j^n = \frac{1}{d_j^\tau} \sum_{i \neq j}^{N_s} \rho_i^n * \eta_{ij}^\tau, \tag{1}$$

where d_j^τ is the number of nodes whose distance from node n_j is no longer than τ hops; N_s is the number of nodes in the substrate network; ρ_i^n is the residual ratio of compute resources capacity at n_i (ratio between available to total resources) and η_{ij}^τ is the residual ratio of bandwidth for the path between n_i and n_j with no longer than τ hops (ratio between available to total bandwidth).

Similarly, the connectivity of substrate links, measures the connectivity of a link based on the number of adjacent links, the residual bandwidth available on these adjacent links, and the compute resources available at the nodes connected by these links. The following equation computes connectivity κ_j^L for any given link L_j in the network:

$$\kappa_j^L = \frac{1}{d_j^L} \sum_{i \neq j}^{L_s} \rho_i^L * \eta_{ij}^n, \tag{2}$$

where d_j^L is the number of adjacent links to link L_j, where adjacent links refer to links that have one common node with; L_s is number of adjacent links in the substrate network; ρ_i^L is the residual ratio of bandwidth for the adjacent link L_i and η_{ij}^n is the residual ratio of compute resources capacity at the common node between link L_i and link L_j.

Both the RFD for nodes, r_j^n, and links, r_j^L, is then computed as the complement of their respective connectivity:

$$r_j^n = 1 - \kappa_j^n, \tag{3}$$
$$r_j^L = 1 - \kappa_j^L. \tag{4}$$

In other words, it measures how disconnected or fragmented a resource is. Naturally, higher RFD indicates a higher degree of fragmentation in the substrate network.

III. Fragmentation-Aware VNF-CPP

In [9], we formulated the VNF-CPP problem as an Integer Linear Programming (ILP) problem where the objective was to minimize resource consumption while satisfying constraints such as processing powers, bandwidth, and latency. To this end, two binary decision variables were introduced; $x_n^{s_k^i}$ that was set to 1 if the VNF s_k^i was allocated on server $n \in N$ and $y_{uv}^{s_k^i s_{k'}^i}$ which was set to 1 if the virtual link between s_k^i and $s_{k'}^i$ was routed over the physical link $(u, v) \in L$. Hence, the following objective function was formulated:

$$\min_{x,y} \sum_{i=1}^{|S|} \Big(\sum_{s_k^i \in S^i} \sum_{n \in \mathbb{N}} M_n x_n^{s_k^i} + \sum_{s_k^i \in S^i} \sum_{s_{k'}^i \in S^i} \sum_{(u,v) \in \mathbb{L}} y_{uv}^{s_k^i s_{k'}^i} \Big). \tag{5}$$

The first term tried to minimize the number of utilized edge servers by using the binary cost M_n. In particular, M_n was equal to 1 if the server n was an edge server ($n \in N_E$) and 0 otherwise.

We modify our previous VNF-CPP problem formulation to include the RFD metric into our ILP objective function, to better guide the placement decisions, targeting both efficient resource utilization and minimum resource fragmentation:

$$\min_{x,y} \sum_{i=1}^{|S|} \Big(\sum_{s_k^i \in S^i} \sum_{n \in \mathbb{N}} M_n r_n x_n^{s_k^i} + \sum_{s_k^i \in S^i} \sum_{s_{k'}^i \in S^i} \sum_{(u,v) \in \mathbb{L}} r_{uv} y_{uv}^{s_k^i s_{k'}^i} \Big). \tag{6}$$

where r_n is node's n RFD, given by Eq. (3), and r_{uv} is link's uv RFD, given by Eq. (4). Including the RFD metric in the objective function ensures that the solution will not only minimize resource consumption and communication costs but will also aim to distribute the VNFs in a manner that leads to lower fragmentation of resources.

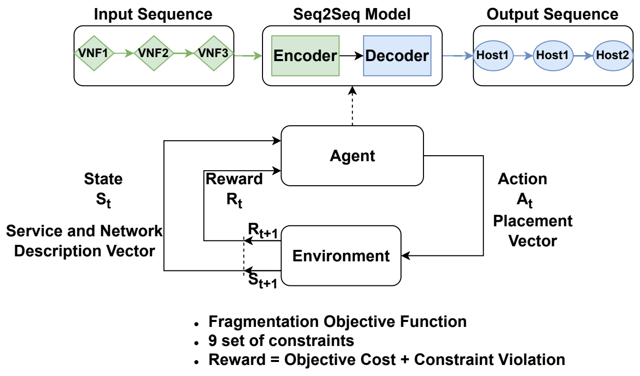

Fig. 2: The RL Agent Environment Interface.

IV. Online VNF Placement using DRL and RCPO

A. Constrained Markov Decision Process Formulation

Following, We model the problem as a Constrained Markov Decision Process (CMDP). Fig. 2 illustrates the interfacing between the RL agent and the environment. State S_t is a vector that describes the requested SFC (m VNFs), including the network state (e.g., available resources), and previously allocated SFCs at time t . An action A_t is a placement vector $p^{S_t} = (p_1, p_2, ..., p_m)$. It represents the decision made by the DRL agent by specifying the placement of the SFC, i.e, the compute nodes hosting the SFC's VNFs. The environment is where the agent's actions get executed. It consists of the ILP formulation of the VNF placement problem, incorporating resource fragmentation of the problem. It also defines the underlying network infrastructure. The reward is the signal the agent receives after taking an action in the environment and it combines the fragmentation objective function cost, and the constraint violation penalty. The fragmentation objective function cost is the ILP objective cost of the VNF placement (Eq. 6). It relates to how fragmented the network resource utilization is after the VNF placements. The constraint violation represents any violations of the predefined constraints (9 sets of constraints [9], [10]). The agent is implemented using a Sequence-to-Sequence (Seq2Seq) neural network consisting of an encoder and a decoder. Moreover, the Bahdanau attention mechanism is used to improve the agent for tasks that involve mapping input sequences (sequence of VNFs) to output sequences (sequence of hosts) [11]. The underlying neural network, denoted by its weights θ, infers a policy $\pi_\theta(p^s \mid s)$ that gives a placement strategy for all possible chains.

B. Solving the VNF placement problem using RCPO

The ultimate goal of the DRL agent is to learn a policy $\pi(\theta)$, defined by the agent's neural network parameters θ, that minimizes the expected cumulative placement cost $J_F^\pi(\theta)$ and also minimizes the expected constraints violation penalty $J_C^\pi(\theta)$ over time while navigating the CMDP. The placement cost is defined by the ILP objective function, whereas the constraint violation penalty is defined for each constraint in the problem formulation. As a result, the primal problem is defined as follows:

$$\min_{\pi \sim \Pi} J_F^\pi(\theta) \quad \text{s.t.} \quad J_{C_i}^\pi \leq 0, \forall c_i \in C. \tag{7}$$

Then, the primal problem is transformed into an unconstrained problem by utilizing the Lagrange relaxation technique, in which the infeasible solutions yield a penalty:

$$\begin{aligned} g(\lambda) = \min_\theta J_L^\pi(\lambda,\theta) &= \min_\theta [J_F^\pi(\theta) + \sum_i \lambda_i J_{C_i}^\pi(\theta)] \\ &= \min_\theta [J_F^\pi(\theta) + J_\xi^\pi(\theta,\lambda)], \end{aligned} \tag{8}$$

where $g(\lambda)$ is the Lagrange dual function; $J_L^\pi(\lambda,\theta)$ is the Lagrangian objective function; λ_i is the Lagrange multiplier (penalty coefficient) for constraint c_i and $J_\xi^\pi(\theta,\lambda)$ is the expected penalization. We solve the following Lagrange dual problem to discover the penalty coefficients which create the best lower bound:

$$\max_\lambda g(\lambda) = \max_\lambda \min_\theta J_L^\pi(\lambda,\theta). \tag{9}$$

We use the RCPO technique to update the Lagrange multipliers automatically. RCPO presents a Lagrange multiplier for each constraint and a penalty term that motivates constraint fulfillment. It then uses a multi-timescale approach that alternates between updating the policy using a policy optimization method such as the Proximal Policy Optimization (PPO) and updating the Lagrange multipliers based on the constraint violations. This process is iteratively repeated until convergence.

Accordingly, RCPO introduces a two-timescale method that involves fast and slow timescales, i.e., θ is calculated by solving Eq. (9) and updating it on the fast timescale, where on the slower timescale, λ is expanded gradually until the constraint violation vanishes. In this two-timescale approach, the policy parameters are updated using a reinforcement learning algorithm to maximize the expected discounted reward. At the same time, the Lagrange multipliers are updated using a slower timescale to enforce the constraint. In order to compute the weights θ that optimize $J_L^\pi(\theta)$, we use the Monte-Carlo Policy Gradients and the gradient descent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J_L^\pi(\theta). \tag{10}$$

Policy gradient techniques select actions directly from an initialized parameterized model and then adjust the model weights so that the subsequent predictions achieve better-expected returns. The log-likelihood approach [12] is then used to compute the gradient of the Lagrangian:

$$\nabla_\theta J_L^\pi(\theta) = \mathop{\mathbb{E}}_{p\sim\pi_\theta(|s)}[L(p \mid s)\nabla_\theta log\pi_\theta(p \mid s)], \tag{11}$$

where $L(p \mid s)$ is computed by summing up the placement cost and the sum of all the constrained violation penalties weighted by the Lagrange multipliers:

$$L(p \mid s) = E(p \mid s) + \sum_i \lambda_i C_i(p \mid s). \tag{12}$$

The gradient is then approximated with Monte-Carlo sampling, as follows:

$$\nabla_\theta J_L^\pi(\theta) \approx \frac{1}{B}\sum_{j=1}^{B} A(p_j \mid s_j)\nabla_\theta log\pi_\theta(p_j \mid s_j), \tag{13}$$

where B is the size of SFCs batch that need to be allocated. $A(p_j \mid s_j) = (L(p_j \mid s_j) - b_{\theta_v}(s_j))$ is called the advantage, where $b(s)$ is a baseline estimator [13]. The baseline is a supporting model that estimates the placement cost plus the penalty received by the agent while following the current policy. This estimator helps reduce variance, accelerate convergence, and improve sample efficiency. It is implemented as an auxiliary sequence network that uses the weights θ_v to parameterize the network. Moreover, it is trained using gradient descent by minimizing the Mean Squared Error (MSE) between its predicted value $b_{\theta_v}(s)$ and $L(p \mid s)$:

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{j=1}^{B} ||b_{\theta_v}(s_j) - L(p_j \mid s_j)||^2. \tag{14}$$

The updated rules for the Lagrange multipliers are given by [7]:

$$\lambda_{k+1} = \Gamma_\lambda\left[\lambda_k - \eta_1(k)\nabla_\lambda J_L^\pi(\lambda_k,\theta_k)\right], \tag{15}$$

$$\nabla_\lambda J_L^\pi(\lambda,\theta) = -\left(\mathbb{E}_{s\sim\mu}^{\pi_\theta}[C(s)]\right), \tag{16}$$

where Γ_λ restricts the value of λ to the range between zero and a maximum value, denoted as $\lambda_{\max}$, by projecting it into this range. The algorithm defines the maximum value of λ and ensures that the optimization process is well-behaved [7].

C. Agent training using DRL and RCPO

Algorithm 1 outlines the training process for the RL agent to solve the VNF Placement problem using DRL and RCPO. The input is a set of SFCs, $\mathcal{S}$, used for training purposes and the number of SFCs in each batch, B. The algorithm initializes the policy parameters θ and θ_v with random weights and sets the Lagrange multipliers λ equal to zero. Then, it iterates through epochs k and performs the following steps in each epoch: i) the gradients $d\theta \leftarrow 0$ are reset; ii) for each SFC in the batch, the SFC description s_j, placement solution p_j, and baseline b_j or the estimate, are sampled, and then the Lagrangian cost $L(p_j \mid s_j)$ is computed using Eq. (12); iii) the gradient of the Lagrangian objective $\nabla_\theta J_L^\pi(\theta)$ using Eq. (13), and the Mean Squared Error (MSE) $\mathcal{L}(\theta_v)$ are computed using Eq. (14); iv) the policy parameters θ and θ_ν are updated using the Adam optimizer (slow learning rate); v) the gradient of the Lagrangian is computed with respect to the Lagrange multipliers $\nabla_\lambda L(\lambda, \theta)$ using Eq. (16), then the Lagrange multipliers λ_i are updated using Eq. (15) (fast learning rate). After iterating through all the epochs, the algorithm returns the updated policy parameters θ and θ_v.

Algorithm 1: Train Agent using DRL and RCPO

```
1  Input: SFC Learning Set $\mathcal{S}$ and SFC Batch $B$
2  Initialize: policy parameters $\theta$ and $\theta_v$ with random
     weights, and Lagrange multipliers $\lambda = 0$
3  foreach epoch $k = 0, 1, 2, \ldots$ do
4      Reset gradients: $d\theta \leftarrow 0$
5      foreach $j \in \{1, \ldots, B\}$ do
6          Sample input $s_j$, solution $p_j$, and baseline $b_j$
7          Compute cost $L(p_j \mid s_j)$ using Eq. (12)
8      Compute $\nabla_\theta J_L^\pi(\theta)$ using Eq.(13)
9      Compute MSE $\mathcal{L}(\theta_v)$, using Eq. (14)
10     Update $\theta$ and $\theta_\nu$ using Adam optimizer (Slow)
11     Compute $\nabla_\lambda L(\lambda, \theta)$ using Eq. (16)
12     Update $\lambda_i$ using Eq. (15) (Fast)
13 Return policy parameters $\theta$ and $\theta_v$
```

D. MinFragRL Heuristic: Inference with Error Correction

Inference is the process by which an agent takes the knowledge learned during training and uses it to make decisions in its environment. The expected penalty when the agent converges is close to zero but not equal to zero. Therefore, theoretically, we should expect the agent to have very few mistakes. To address this problem, we propose a heuristic that complements the solution from the RL agent with an error correction mechanism; this heuristic is named MinFragRL.

MinFragRL's flowchart is illustrated in Fig. 3. First, a state vector is created for the incoming SFC request. This state vector represents the current network state and the requirements of the incoming SFC request in a format suitable for the RL agent. The state vector is then passed to the RL Agent. Using its learned policy, the RL agent produces a

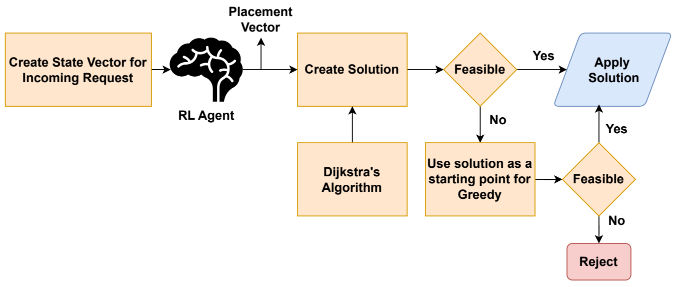

Fig. 3: MinFragRL flowchart.

placement vector, which tells where to place each VNF of the SFC requested. Then, the well-known Dijkstra's algorithm is used to find the shortest paths interconnecting the SFC's VNFs. Therefore, using the placement vector and Dijkstra's algorithm, an initial solution for the requested SFC is devised based on the RL agent's recommendation. Before applying the solution, it undergoes a feasibility check. This step ensures the proposed solution is viable and does not violate any constraints.

If the solution is deemed feasible, it is applied straight away. Otherwise, an error correction mechanism is triggered. The initial infeasible solution is used as a starting point for a greedy method. The greedy method tries to incrementally improve in a step-by-step manner the infeasible solution to find a feasible solution. For example, if one of the hosts chosen by the initial solution cannot be used, the greedy method replaces this host with the next available host. If no other hosts can be used, then it fails to correct the error. The greedy method is fast and simple, but it does not provide a guarantee to correct all the errors in the initial infeasible solution. Therefore, another feasibility check is then performed on the improved solution. If the refined solution is feasible, it is applied. If it is still not feasible after the error correction mechanism, the request is rejected. The main idea of MinFragRL is to leverage the strength of an RL agent to provide initial VNF placements but with a backup plan.

V. Results and Discussions

This section shows the evaluation results of using MinFragRL Heuristic. For this purpose, the RL agent is trained to place 100 SFC requests and the training runs for 30,000 epochs. We used a homogeneous network consisting of 11 compute nodes, and 2 routing nodes interconnected using a star topology (Fig. 4). To train the agent, we used a Google Cloud Virtual Machine, which has a system RAM of 83.5 GB, GPU A100 (40GB version) and 166.8 GB of storage. Training time was 80 minutes on average.

Fig. 5 shows the costs incurred during the agent's training phase on the star network topology. We remind that the fragmentation cost represents the fragmentation degree in the network that results from the agent's VNF placement decisions. The better the VNF placement, the lower the fragmentation degree. As we observe, minor fluctuations are experienced during the initial epochs. After that, it steadily

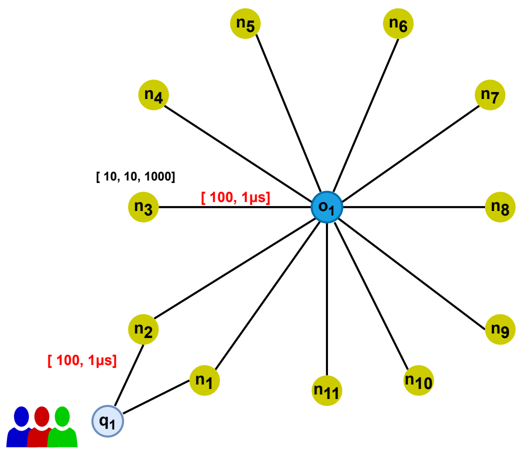

Fig. 4: Overview of the star topology network.

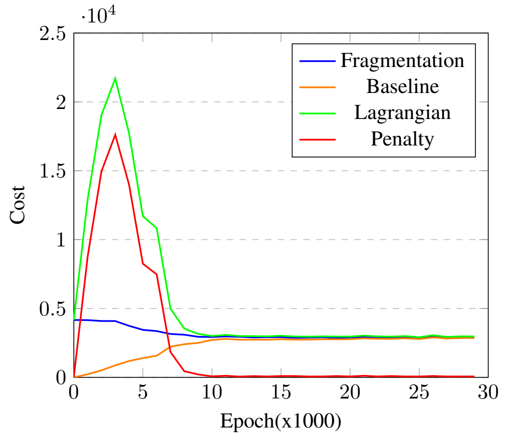

Fig. 5: Costs during training.

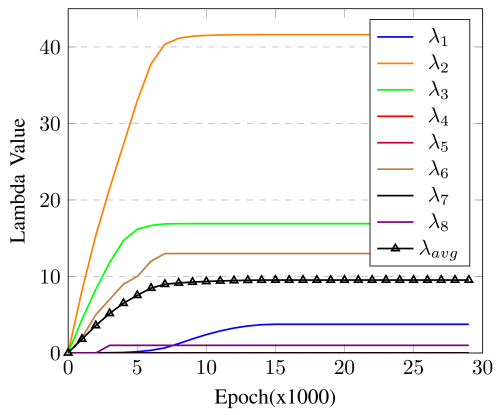

Fig. 6: Lambda values during training.

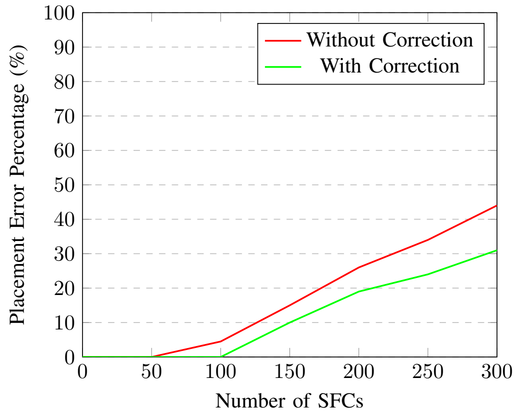

Fig. 7: Placement errors.

decreases until it converges to its lowest value. On the other hand, the baseline cost, the estimate of the fragmentation cost, starts from a very low value and then rapidly increases until it converges to a value close to the fragmentation cost. The penalty cost, which represents the constraint violation cost, sharply rises to its peak and then experiences a rapid decline and plateaus at the convergence. Finally, the Lagrangian cost, which is the sum of fragmentation and penalty costs, converges to a value very close to the fragmentation cost.

Fig. 6 shows the values of the Lagrange multipliers during the training for the start network, along with their average. In this formulation, we identify nine types of constraints [9], [10]. Therefore, there are nine Lagrange multipliers whose values are determined via training. The Lagrange multipliers values rise to certain levels and stabilize at convergence, indicating they have reached their optimal or desired values.

Going back to Fig. 5, we observe that after convergence, the penalty cost oscillates slightly. This will result in placement errors from the agent and that is why it is important to complement the outcome of the RL agent with an error correction mechanism to mitigate this phenomenon. Fig. 7 visually represents the placement error percentage in relation to the number of SFCs requested. The red line represents placement errors without correction. The line starts at 0% error when the number of SFCs is 0 or 50. As the number of SFCs increases, so does the placement error percentage. By the time there are 300 SFCs, the error is at 44%. On the other hand, the green line signifies placement errors with correction. This line starts with no errors for up to 100 SFCs. From this point, the error increases slower than the without correction line. By the time there are 300 SFCs, the error is at 31%.

The agent was trained to place up to 100 SFC requests, but it failed to allocate all the 100 SFCs as it provided the wrong placements for 5 SFCs out of 100. However, with the use of a simple error correction mechanism suggested in our heuristic MinFragRL, we can allocate all 100 SFCs correctly. We also note here that the agent was trained to place 100 SFCs, but we also tried to use it to place more than 100 SFCs, as indicated in the graph. In this case, the agent uses only its knowledge of placing up to 100 SFCs, and it is expected that we may have more errors that cannot be corrected.

Fig. 8 compares our proposed heuristic against the ILP method (Global Optimal), SeqSort [9], and VNE-RFD [8] in terms of fragmentation cost, which is the objective cost of Eq. 6, per number of SFCs. Fig. 9 compares these methods in terms of success ratio against the number of SFCs to be

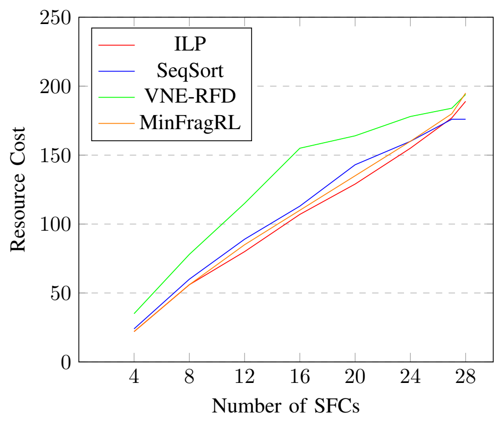

Fig. 8: Resource cost per number of requested SFCs.

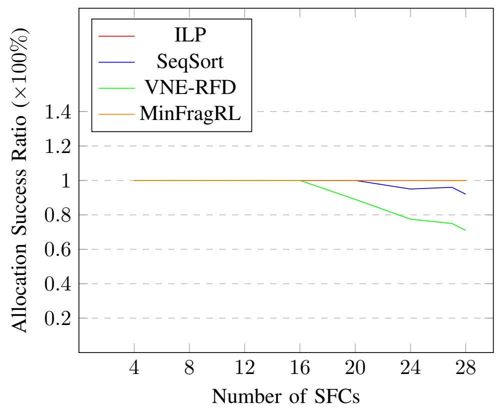

Fig. 9: Allocation success ratio per number of requested SFCs.

placed. SeqSort is a sequential heuristic that sorts all SFCs and then allocates them individually. VNE-RFD is an online placement algorithm that considers the RFD metric too. It is important to consider both Fig. 8 and Fig. 9, as these two figures complement each other. In Fig. 8, all methods show an increase in cost as the number of requested SFCs grows. The ILP and MinFragRL methods have identical costs for 4 and 8 SFCs but diverge for more. By looking at Fig. 9, we can conclude that VNE-RFD fails to allocate all SFCs when the number of SFCs exceeds 16 and SeqSort fails to allocate all SFCs when the number of SFCs exceeds 20. Interestingly, MinFragRL is the only heuristic in this comparison that was able to allocate all the SFC requests the same as the global optimal method, ILP.

VI. Conclusion

This paper presented a fragmentation-aware VNF placement, utilizing a combination of DRL and Reward Constrained Policy Optimization (RCPO), enabling adaptive learning of Lagrange multipliers for constraint satisfaction. Our experimental study showcased the method's superiority in terms of allocation success ratio, average placement cost, and constraint violation penalty compared to state-of-the-art methods. The results confirmed that our approach presents improved adaptability, resource efficiency, and constraint satisfaction capabilities. As future work, we plan to extend the proposed method to address more complex network scenarios and constraints and explore further integrating other optimization techniques to improve its performance. Additionally, another promising direction could be investigating the potential of transfer learning and meta-learning approaches to enhance the DRL-RCPO framework's ability to adapt to new network environments.

References

[1] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-efficient vnf placement and scheduling in public cloud networks," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 4946–4959, 2020.

[2] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[3] D. Qi, S. Shen, and G. Wang, "Towards an efficient vnf placement in network function virtualization," *Computer Communications*, vol. 138, pp. 81–89, 2019.

[4] Z. Chen, H. Li, K. Ota, and M. Dong, "Deep reinforcement learning for aoi aware vnf placement in multiple source systems," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 2873–2878.

[5] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2019.

[6] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[7] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[8] H. Lu and F. Zhang, "Resource fragmentation-aware embedding in dynamic network virtualization environments," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 936–948, 2022.

[9] R. Mohamed, A. Leivadeas, I. Lambadaris, T. Moris, and P. Djukic, "Online and scalable virtual network functions chain placement for emerging 5g networks," in *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE, 2022, pp. 1–6.

[10] ——, "Fast resource allocation for virtual network functions chain placement," in *2022 International Telecommunications Conference (ITC-Egypt)*. IEEE, 2022, pp. 1–6.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Reinforcement learning*, pp. 5–32, 1992.

[13] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.