A Fair Approach to the Online Placement of the Network Services over the Edge

Masoud Taghavian*[†], Yassine Hadjadj-Aoul*[‡], Géraldine Texier *[†], Nicolas Huin [†], Philippe Bertin*[§]
*IRT-BCOM, France; firstname.lastname@b-com.com
[†]IMT Atlantique/IRISA/Adopnet, France; firstname.lastname@imt-atlantique.fr
[‡]University of Rennes, Inria, CNRS, IRISA, France; firstname.lastname@irisa.fr
[§]Orange, France; firstname.lastname@orange.com

***Abstract*—The unavoidable transition from rigid dedicated hardware devices towards flexible containerized network services, introduced by Network Function Virtualization (NFV), brings novel opportunities while presenting several new challenges. Indeed, meeting the expectations of NFV in post-5G networks depends on the efficient placement of the services. The online placement of network services, demanding strict end-to-end latency requirements, with restricted computing resources presents a challenging problem which is worth investigating. We propose a Branch-and-Bound search approach for finding optimal placements of the network services by applying several cost functions to maximize the service acceptance. Extensive evaluations have been carried out, and the results confirm significant improvements when we consider a fair distribution of the resources on the edge.**

***Index Terms*—Network function virtualization, Placement, Branch-and-Bound, Edge, QoS.**

I. Introduction

Network operators adopt Network Function Virtualisation (NFV) and Software-Defined Networking (SDN) in 5G networks to create more flexible networks that can rapidly integrate new services. In NFV and SDN, service placement is a crucial step that involves allocating heterogeneous network resources for service requests meeting Quality of Service (QoS) constraints. In terms of complexity, the problem of the placement falls into the class of NP-Complete [1]. It is encountered in various NFV uses-cases in post-5G networks, ranging from Virtual Network Function Forwarding-Graph (VNF-FG) placement and Network Slicing to virtualization of the Internet of Things (IoT), Content Delivery Network (CDN), and Core Network (CN), following many objectives and constraints including the resources, QoS requirements, energy consumption, Revenue to Cost (R2C), Service Acceptance (SA), etc. [2]. We are curious about improving the SA as our objective which is defined as the number of services that can be placed over the network subject to the resources and QoS requirements.

Two general categories considered for the placement problem in the literature are *offline* and *online* placements. In an offline placement scenario, we know all the service requests in advance, while in an online placement scenario, we do not have any information about future service requests, and we need to make a placement for a service request as soon as it arrives (we may also have time restrictions for performing the placement) [2]. In offline placement, by a well-defined Integer Linear Programming (ILP) model, we can perform an optimization to find the maximum number of accepted services (*a.k.a.* acceptance ratio). In online placement, although we do not know the service requests beforehand (consequently, we can not optimize directly the number of accepted services), we can ensure that the placement of the current service request enhances the chances of accepting future service requests. This can be done by optimizing a cost function consistent with our objective. The online placement of the network services, demanding strict End-to-End (E2E) latency requirements, over the edge networks, with restricted computing resources presents a challenging problem worth investigating. To meet the required latency of the network services, we can not place them on the clouds far from the end users, where the resources are abundant. Some services must necessarily stay on the edge, where we are not generous in spending these resources.

Even though the problem of the placement has been investigated in several related areas, to the best of our knowledge, the placement of the service requests with strict E2E latency on the edge network having limited node resources, considering user-location and anti-affinity is not yet addressed.

Generally, the proposed placement approaches optimize the cost of the resources or QoS requirements like bandwidth and latency. A network service requires node resources (*a.k.a.* computing resources) for deploying its Virtual Network Functions (VNFs) and link resources (*a.k.a.* network resources) for realising its Virtual Links (VLs). The optimization involves placing the VNFs as close as possible to the user-location and the other VNFs, to minimize the bandwidth usage and/or the E2E latency. We show that this optimisation approach can have a devastating effect on edge networks with limited node resources, it can lead to draining the resources in the proximity of the end users, which results in service rejection. Accordingly, we do not follow an optimization approach toward the bandwidth/latency, however, we consider them as constraints that need to be satisfied to maximum SA.

The contributions of the current work can be summarized as follows. We model the problem in ILP and resolve the model to obtain the global optimum (which can only be achieved in an offline scenario). We propose an efficient approach based on Branch-and-Bound (BnB), capable of achieving optimal and near-optimal results according to several cost functions and search strategies. We explore several cost functions and demonstrate the inconsistency of bandwidth and/or latency optimization, and the benefits of fair placement, to maximize SA.

The *fairness* of our approach can be explained in two ways. Our achieved improvements are due to the fair distribution of the resources in our placements. On the other hand, our approach makes a fair compromise between the service provider and the network/cloud provider. The service provider prefers that the realized latency of their services be the lowest possible (ideally zero), whilst the network provider would rather it to be the highest possible, allowing them to push the placement of the services from the edge to the clouds where the resources are abundant and cheaper. The maximum acceptable latency for the realized services is where we can work to meet the expectations of both sides.

This paper is organized as follows. First, we explore the related works in Section II. We present our ILP and Column Generation (CG) formulation of the problem in Section III. Then, we provide a comprehensive description of our BnB approach in Section IV. In Section V, we explore the evaluations to assess the efficacy of our proposed solution. We will conclude in Section VI.

II. Related Work

Studying the placement problem in existing research begins with the advent of NFV. It is frequently dealt with in different use cases, considering different categories of the objectives.

A substantial portion of the studied placement approaches includes the Linear Programming (LP) solutions, where the problem is formulated mathematically in ILP or Mixed Integer Linear Programming (MILP).

In [3], the placement is investigated for the network services under multiple constraints to maximise the SA, initially by suggesting an ILP, and subsequently a heuristic. They consider bandwidth and latency for the links, and computing resources for the nodes on a star-centric edge/cloud network topology. Although strict requirements of E2E latency of the service requests and the anti-affinity rules are not studied, limited node resources were considered on the edge, and they demonstrated that a balanced allocation of the resources can improve the number of accepted services.

Jin *et al.* try to address VNF placement problem in [4], considering the resource shortage on the edge with latency guarantees. First, they formulate the problem in MILP to minimize the consumption of the computing resources (by sharing and reusing the already placed VNFs), and the bandwidth. Note that using an already placed VNF reduces the cost of placing a new VNF, but it may result in using more bandwidth and latency to access that VNF. Although they did not consider anti-affinity rules, they proposed an interesting Depth-First Search (DFS) based algorithm for placing the VNFs and the VLs to obtain near-optimal solutions for an online placement scenario.

The placement problem on the edge is also investigated in [5], using MILP over batch-based service requests, minimizing the overall network latency. They show that their model can improve the acceptance rate of the services with strict latency requirements, via sharing the already placed VNFs and considering the affinity rules. They define an affinity matrix to identify the VNFs with high-affinity, that can be placed over the same network node. High-affinity is defined when two VNFs exchange a big amount of data flow, while low-affinity or anti-affinity is considered to allow critical VNFs to be placed on separate network nodes (in case of failure). They proposed a heuristic placement algorithm for big networks and a large number of requests, and they compared it with a Random-Fit algorithm.

Even though the ILP-based methods guarantee optimality, they could face challenges related to scalability. Performing an exact approach could require a long execution time for achieving an optimal result, which could make restrictions for using in large-scale networks. This constraint poses a significant obstacle for online placements which may have constraints on the execution time.

Heuristic approaches are often retained when it comes to scalability. Best-Fit is one of the most popular placement methods, which places the VNF by sorting the nodes according to their resources and placing the VNFs in an iterative way. [6] studies heuristic methods (including the Best-Fit), and the authors propose an algorithm based on a multiple-level graph to find near-optimal results in a way that can scale well to the bigger networks. They evaluate the execution time, the acceptance ratio, and the average placement cost regarding the assigned resources.

In [7], an adaptive heuristic approach is proposed to maximize the total throughput of accepted requests in an edge/core cloud network, considering anti-affinity rules. They try to avoid VNF consolidation to avoid severe performance degradation (*a.k.a.* VNF interference), which is intolerable for some QoS-sensitive 5G use cases (e.g., autonomous driving and 4K/8K HD video).

In spite of being able to achieve the results as quickly as possible, heuristics do not offer the quality. Meta-heuristic and evolutionary algorithms represent a different direction to solving the placement problem. In [8], a fast sub-optimal Tabu Search based approach is proposed to minimize the end-to-end latency and the overall deployment cost. Although they are effective at overcoming local optima, these algorithms often are hindered by unpredictability, particularly regarding execution time, which is of utmost importance in online placement.

Sophisticated AI search algorithms and innovative breakthroughs in Machine Learning (ML) (especially in Deep

Table I
Notations

Name	Description
$G = (V, E)$	Substrate network
V	Set of nodes of the network
E	Set of links of the network
$\omega^+(v)$	Outgoing neighboring nodes of $v \in V$
$\omega^-(v)$	Ingoing neighboring nodes of $v \in V$
B_e	Bandwidth available on the directed link from $v \in F_k$ to $j \in w^+(v)$
$H_k = (F_k, L_k)$	Virtual graph of service k
F_k	Set of VNFs to be placed for service k
L_k	Set of VLs between the VNFs of service k
$\omega^+(f)$	Outgoing neighboring VNFs of $f \in F_k$
$\omega^-(f)$	Ingoing neighboring VNFs of $f \in F_k$
B_l	Bandwidth requested between the two VNFs of $l \in L_k$
C_f	Computing resources requested by $f \in F_k$
C_v	Computing resources available at $v \in V$
D_e	Latency experienced on link $e \in E$
D_k	Latency requested for service $k \in K$
D_l	Latency requested between the two VNFs of $l \in L_k$

Reinforcement Learning (DRL)) have recently caught the interest of the community. An ILP placement approach is proposed in [9] for low-latency IoT services on Multi-Tier Mobile Edge Networks to maximize the throughput and the SA ratio. They devise a Reinforcement Learning (RL) approach to address the online placement of the requests, but they do not take into account anti-affinity rules.

III. The placement problem with fixed user locations

The placement problem is defined as placing a service request graph on an Substrate Network (SN) graph. A service graph $k \in K$, where K is the set of services, is indicated by a directed graph $H_k = (F_k, L_k)$, containing a set of VNFs F_k, and a set of VLs (connecting the VNFs) L_k, plus Service Level Agreements (SLAs) (meeting the QoS requirements). A VNF $f \in F_k$ requests a subset of resources (*e.g.,* CPU C_f), and each VL $l \in L_k$ demands an amount of bandwidth B_l. Likewise, a SN is represented by a directed graph $G(V, E)$ of its nodes V and links E. A node $v \in V$ has a resource capacity (C_v), and a link $e \in E$ has a bandwidth capacity of B_e, besides QoS parameters (*e.g.,* E2E latency). Finally, we consider that a VNF $f \in F$ can only be instantiated on a subset of nodes $V_f \subseteq V$ of the SN. In the case of a user location, this subset can be reduced to only one node. The problem is finding the mappings of the VNFs and the VLs of the service requests into the network nodes and the network paths respectively, subject to the constraints (Table I represents notations).

Affinity and anti-affinity rules are used in cloud computing for creating a balance between the performance and reliability of the placed service. Affinity proposes placing the VNFs on the same network node for improving inter-networking performance. While anti-affinity is employed to enhance the reliability and availability by avoiding certain VNFs of a service from using the same physical resources in an effort to decrease the consequences of a network node failure [10]. Moreover, placing the VNFs of the same service over the same network node (*a.k.a.* VNF consolidation) may cause severe performance degradation (*a.k.a.* VNF interference), which is not acceptable for some QoS-sensitive 5G use cases [11].

We consider anti-affinity for all the VNFs of the service (*i.e.,* all of the VNFs of a service request are placed over separate network nodes), since skipping this limitation leads to a substantial relaxation of the problem such that the gain of using different placement approaches becomes marginal.

A. Embedding Decomposition

Similarly to our previous work [12], we exploit a decomposition method for solving the offline placement problem. We still consider the *embedding decomposition*, where each variable represents a possible embedding of the service onto the physical network. The main difference relies on the fixed user location, which slightly impacts the pricing problem. Once again, the number of embeddings is exponential and an embedding-based formulation demands an exponential number of variables (columns). Nevertheless, a solution consists of only a few of these variables. To create only useful variables, we use the CG algorithm. CG is used for solving large-scale linear programs with the help of a back-and-forth resolution of a *master problem* and *pricing problems.* Beginning from a *reduced* master problem, the master problem gives dual values to the pricing problems, and the pricing problems give improving columns to the master problem. For every service k, we have to create an embedding γ among all possible embeddings Γ_k.

1) Master problem: Only the set of variables $z_{k\gamma} \in \mathbb{N}$ is needed to represent the number of services k allocated on the embedding γ. Every embedding γ is determined by the amount of bandwidth it requires from every link e, as $\delta_e(\gamma)$, and the number of CPUs required from every node v, as $\theta_v(\gamma)$.

For every service $k \in K$, we limit the number of embedded with the constraints

$$\sum_{\gamma \in \Gamma_k} z_{k\gamma} \le n_k, \tag{1a}$$

where n_k is the number of requests requiring the same service k.

For each node $v \in V$, we define its capacity constraints as:

$$\sum_{k \in K} \sum_{\gamma \in \Gamma_k} \theta_v(\gamma) z_{k\gamma} \le C_v. \tag{1b}$$

And for each link $e \in E$, we define its capacity constraints as:

$$\sum_{k \in K} \sum_{\gamma \in \Gamma_k} \delta_e(\gamma) z_{k\gamma} \le B_e, \tag{1c}$$

The objective function can be written as:

$$\max \sum_{k\in K}\sum_{\gamma\in\Gamma_k} z_{k\gamma}. \tag{1d}$$

2) Pricing problems: Given a service $k \in K$, we formulate the corresponding embedding problem as an ILP.

- $x_e^l \in \{0,1\}$ indicates if the virtual link $l \in L_k$ of the service is routed through the physical link $e \in E$.
- $y_v^f \in \{0,1\}$ indicates if the VNF $f \in F_k$ of the service is instantiated in node $v \in V_f$.

The following set of constraints ensures that a VNF $\forall f \in F$ is embedded on a physical node:

$$\sum_{v\in V} y_v^f = 1. \tag{2a}$$

The following set of constraints ensure that a node $v \in V$ can host up to one VNF of the service:

$$\sum_{f\in F_k} y_v^f \leq 1 \tag{2b}$$

We ensure flow conservation for each node $v \in V$ and each VL $l = (f_i, f_j) \in L_k$ with:

$$\sum_{e\in\omega^(v)} x_e^l - \sum_{e\in\omega^-(v)} x_e^l + y_v^{f_i} - y_v^{f_j} = 0. \tag{2c}$$

The overall latency of the service is ensured with:

$$\sum_{e\in E} D_e \sum_{l\in L_k} x_e^l \leq D^k, \tag{2d}$$

and the latency between each VL $l \in L_k$ with:

$$\sum_{e\in E} D_e x_e^l \leq D_k \tag{2e}$$

3) Pricing objective function: The pricing problems generate improving columns for the master problem. Improving and non-improving columns are different in their reduced costs, as the reduced cost of a variable represents the improvement of the objective function if the variable is present in the solution. The pricing problem aims to find the columns with the best reduced cost. If all variables have a null reduced cost, then the CG algorithm has converged.

We can get a variable's reduced cost formula from the dual of the master problem. If we define by π the dual values corresponding to the constraints of the primal problem (and let the exponent define the corresponding constraint), the dual problem is formulated as:

$$\min \quad \sum_{k\in K} n_k \pi_k^{(1a)} + \sum_{e\in E} B_e \pi_e^{(1c)} + \sum_{v\in V} C_v \pi_v^{(1b)} \tag{3a}$$

$$s.t. \qquad \pi^{(1a)} + \sum_{e\in E} \delta_e(\gamma)\pi_e^{(1c)} + \sum_{v\in V} \theta_v(\gamma)\pi_v^{(1b)} \geq 1 \qquad \forall k \in K, \forall \gamma \in \Gamma_k \tag{3b}$$

Columns in the master problem turn into constraints in the dual problem. Similarly to the CG algorithm, the row generation algorithm begins from a reduced problem and looks for unsatisfied constraints. Given a dual solution $\bar{\pi}$, the separation problem of the dual searches an embedding that violates constraints (3b), *i.e.,* any embedding γ such that

$$\bar{\pi}^{(1a)} + \sum_{e\in E} \delta_e(\gamma)\bar{\pi}_e^{(1c)} + \sum_{v\in V} \theta_v(\gamma)\bar{\pi}_v^{(1b)} < 1. \tag{4}$$

Returning to the embedding sub-problem for a given service k, defined by constraints (2), the objective function becomes

$$\min \sum_{e\in E} \bar{\pi}_e^{(1c)} \sum_{l\in L_k} B_l x_{el} + \sum_{v\in V}\sum_{f\in F_k} C_f \bar{\pi}_v^{(1b)} y_{vf}. \tag{5}$$

If the optimal value of this problem is strictly less than $1 - \pi_k^{(1a)}$, we know that adding the corresponding embedding into the master problem will improve the solution. Otherwise, no embedding can improve the master problem.

IV. Proposed Solution

In the previous section, we addressed our placement problem using CG. We can obtain the global optimum (*i.e.,* the maximum feasible number of placed services) with CG in an offline placement scenario. But, when it comes to the online placement, since we do not know all the service requests in advance, we need to ensure that the placement of the current service request improves the possibility of accepting future service requests. This can be done by optimizing a cost function consistent with our objective. For simplicity, we utilise the terms *network* and *service* for the graphs of the Physical Network and the Virtual Service.

BnB is one of the most widely used paradigms of designing algorithms for solving optimization problems with exponential complexity. Many types of constraints exist in the placement problem, which can be satisfied in BnB in an efficient way (the more constraints we have, the more we can prune the tree, resulting in a faster search procedure). Fig. 1 represents our BnB search tree. We call a node of the search tree as a *state* to differentiate it from a network node. Each state carries placement information, as well as a complete image of the network with its resources and QoS metrics. The terminal states include a complete placement of a service over a network.

Expanding States: Beginning with the initial state of the tree, we choose a VNF from the service and create the associated sub-states for each network node that can supply the required resources (a parent-child relationship). A state includes a snapshot of the entire network after the placement of the VNF on the associated node. After the placement of a VNF, we place its connected VLs under the condition that their source and destination VNFs are already placed. To place a VL, we look for a shortest path from the node hosting the source VNF to the node hosting the destination VNF, taking into account the required bandwidth and latency. After placing a VLs, the network image of the state is updated accordingly.

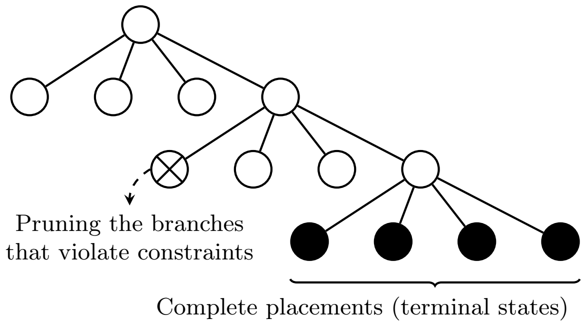

Figure 1. A sample BnB search tree, placing a service with 3 VNFs over a network with 4 nodes

VNFs' Selection: At each state, we need to select a VNF to place it over the network. For this selection, we need an ordered sequence of the VNFs. To create this list, we make a Breadth-First Search (BFS) traversal over the service, beginning from the service entry VNF (BFS traversal guarantees a connected sub-graph of the service on every state, representing a partial placement that we can be verified to meet our constraints). The depth of the corresponding state indicates the position of the VNF in the list of the VNFs to be selected for the placement. The search tree grows by continuing to choose a VNF from the service and to place it on the network nodes, creating their associated sub-states. If a service of N VNFs is placed on a network of M nodes, we would have a search tree which will grow to the depth of N with a branching factor of M.

Cost Function: The cost function evaluates the states to determine which state to expand at each iteration of the search procedure. It is served as a measurement to allow comparing different states. Since the states represent different partial or complete placements of the service (they may have placed more or fewer VNFs and VLs), we need to ensure that the proposed cost function can compare all of the states.

Search Procedure: The search procedure is an iterative procedure involving a sequence of the states, which we call *fringe*, and only contains the initial state at the beginning of the search. At each iteration, we pop and expand the head state from the fringe, and we store the generated sub-states in the fringe. The search procedure continues the iterations until we reach a terminal state or we exceed a timeout (failure). After expanding a state, we verify that the sub-states respect the constraints (concerning resources, latency, and anti-affinity). We only store the non-violating sub-states in the fringe (*i.e.,* others are discarded). The sequence of the states kept in the fringe specifies the traversal or the direction of the search. To traverse the search tree in DFS, the sub-states of the current state are evaluated and sorted by the cost function, then they are pushed into the fringe (which is a stack in DFS) in order so that the head state (that will be popped for the expansion) has the minimum cost. In Uniform-Cost Search (UCS) traversal, the sub-states of the current state are evaluated by the cost function and they are added to the fringe (which is a list in UCS). The fringe in the UCS traversal is a sorted list, thus the head state has the minimum cost among all of the states, *i.e.,* the UCS traversal is complete and optimal.

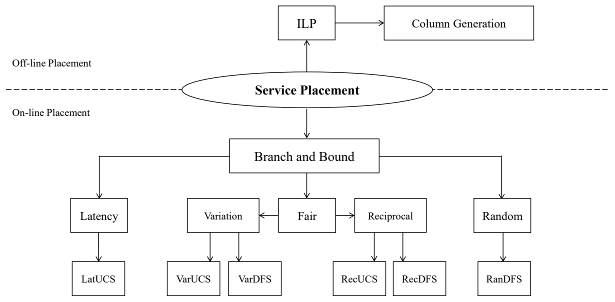

Figure 2. Overview of the proposition

Search Strategies: A search strategy includes a cost function and a search traversal that can be applied over our BnB formulation. We propose several search strategies, by following three main approaches (Fig. 2 demonstrates a taxonomy of the proposed strategies). (1) *Latency Optimized Placement*: We propose the LatUCS search strategy, which performs UCS traversal over a cost function defined as the sum of the latencies of the realized paths that place VLs. The idea is to investigate the effect of latency optimization on the objective of SA. (2) *Random Placement*: We propose the RanDFS search strategy, which performs a DFS traversal over a cost function that returns a random number. The idea is to investigate a worst-case random placement on the objective of SA. (3) *Fair Placement*: We propose several search strategies to realize a fair distribution of the node resources. *VarUCS and VarDFS*, which perform UCS and DFS traversals over a cost function defined as the variance of the available resources over all network nodes. *RecUCS and RecDFS*, which perform UCS and DFS traversals over a cost function defined as the average of the reciprocal function of the available resources (+1 is added to prevent division by zero) over the network nodes that place VNFs ($\overline{\frac{1}{r+1}}$). We use the cost function proposed in [3], which tries to put a high cost for a placement on a network node which is short on resources.

V. Experimentation

Our implementations are made in Java, and we use the Gurobi solver. All of our experimentations are made on a PC with a Core-i7 CPU and 8GB of RAM. To evaluate our search strategies following our objective of SA, we begin by initializing the resources of the network nodes and links. Next, we repeatedly create a service request according to the specified parameters, and then place it on the network using the specified strategy. If we accomplish placing the requested service, we carry out the placement

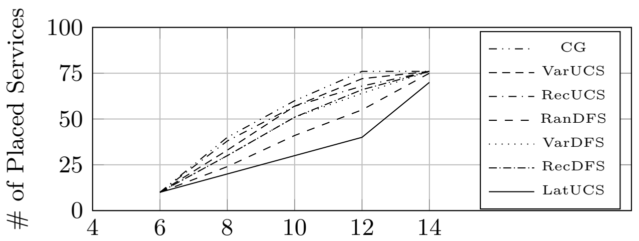

Figure 3. Number of placed services comprised of 3 VNFs requiring different E2E latencies

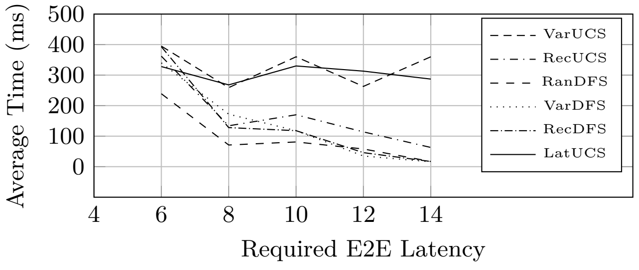

Figure 4. Average time of placing services comprised of 3 VNFs requiring different E2E latencies

by reserving its required resources on the network, and we begin a subsequent iteration. If we fail to place the service request, we terminate the evaluation.

We perform our evaluations on the BT-Europe, BT-North-America and Grid-7x6 topologies (chosen from Zoo Topology dataset), with respectively 24, 36 and 42 nodes, and 74, 152 and 142 links. We consider 10 units of CPU for each network node and 1 unit of latency for each network link, and daisy chain topology for the service graphs (representing the topology of Service Function Chains (SFCs)). Each service contains between 3 to 5 VNFs with associated VLs, and each VNF requires one unit of CPU. The user location is fixed at the network node *12* for BT-Europe, *34* for BT-North-America, and *0* for Grid-7x6. The E2E latency is calculated as the sum of the latencies of the paths that place the VLs, starting from the user location and passing through each VNF and returning to the user location.

A. Latency Optimization

Latency is a crucial constraint that should be set carefully. If a service requires the minimum feasible latency, we may not have many choices for the placement and it would only be as close as possible to the user location. Similarly, if a service requires a high E2E latency, so that we can place it almost wherever, regardless of the selected search strategy, we will end up placing almost the same number of service requests until we drain all of the network node resources.

Fig. 3 represents the number of placed services comprised of 3 VNFs requiring different E2E latencies over BT-Europe network. As shown in Fig. 3, all of the search strategies place almost the same number of services for the lowest

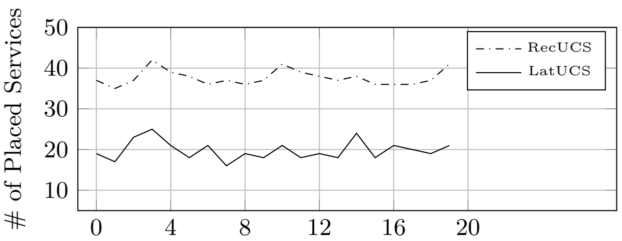

Figure 5. Number of placed services in each random experiment

feasible latency (6 units), and the high-enough latency (14 units). Considering the E2E latency of 8 to 12 (called *effective* range), LatUCS places the minimum number of services, and CG has the maximum placements. Note that the number of placed services found by the CG is on average almost twice (exactly 1.96 times, or 96% improvement) the number of placements by LatUCS. Even RanDFS can place more services than LatUCS by making random placements (1.31 times more on average).

1) Random Service Requests: Until now, we set a constant size for the service requests throughout an evaluation. But, in practice, the service requests can have mixed sizes, resource requirements, and latency requirements. To have realistic experiments, we need to create random services and repeat these experiments many times with multiple seeds of randomness. For every random experiment, we begin by creating a service of a random size in a range of [3, 5], with a random latency in the *effective* range based on its size. We proceed with the placements as long as the chosen strategy does not fail. As shown in Fig. 5, by comparing RecUCS, which performs a fair placement, with LatUCS, which perform latency optimization, we witness that on average we can place 1.9 times more services using RecUCS instead of LatUCS.

B. Fair Placement

Although VarUCS and RecUCS achieve almost the same results by placing 1.78 and 1.83 times more services than LatUCS on average, their execution times are considerably different. Fig. 4 show the average execution time of placing the services comprised of 3 VNFs requiring different E2E latencies over the BT-Europe network. As it is shown in Fig. 4, RecUCS can place the services 1.95 times faster on average than VarUCS. In addition, as we consider the average execution time of placing the services comprised of 4 VNFs, the execution time of VarUCS grows significantly by increasing the E2E latency. In RecUCS's case, the execution time decreases similarly to the DFS-based strategies, which shows the scalability of RecUCS.

C. Big Picture

1) Service Acceptance: Exhaustive evaluations were performed, considering both networks of BT-Europe, BT-North-America and Grid-7x6 with service requests of different sizes (3 to 5 VNFs), requiring an *effective* range of

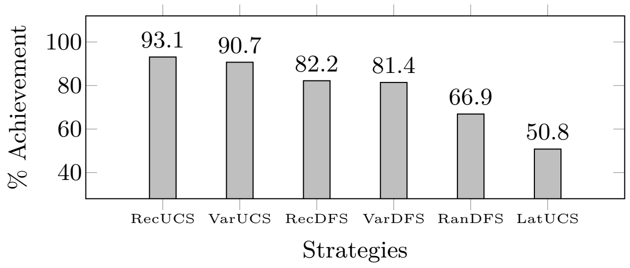

Figure 6. Average achievement to the global optimal results by our online strategies

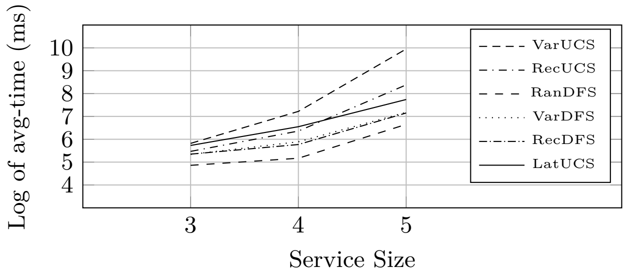

Figure 7. Logarithm of the average placement time for placing different service sizes on BT-Europe network

E2E latency, and user-location fixed on different network nodes. Fig. 6 shows the average percentage of the number of placements achieved by our proposed online placement strategies in comparison to our CG results.

2) Execution Time: Since the execution time of our strategies grows exponentially by increasing the service size, we represent the logarithm of the average placement execution time in Fig. 7. The complexity of our approach depends on the growth of expanded states, which is determined by the search strategy. While the complexity of our strategies grows exponentially, they do not grow equally. If we can sort the growth of the complexity of our strategies, we can put VarUCS in the first place. RecUCS is placed between VarUCS and RecDFS/VarDFS (which have almost similar growth), and finally RanDFS.

VI. Conclusion

The placement problem has been studied for several recent years, and it is one of the most popular topics in NFV because of the constantly developing use-cases with evolving requirements and constraints. In this paper, we studied the online placement of constrained services on edge networks, and we proposed an efficient approach, capable of achieving 93% of the global optimum which can only be achieved in on offline placement.

References

[1] Z. Chen, S. Zhang, C. Wang, Z. Qian, M. Xiao, J. Wu, and I. Jawhar, “A novel algorithm for NFV chain placement in edge computing environments”, in *2018 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2018, pp. 1–6.

[2] G. Mirjalily and L. Zhiquan, “Optimal network function virtualization and service function chaining: A survey”, *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 704–717, 2018.

[3] C. Morin, G. Texier, C. Caillouet, G. Desmangles, and C.-T. Phan, “VNF placement algorithms to address the mono-and multi-tenant issues in edge and core networks”, in *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*, IEEE, 2019, pp. 1–6.

[4] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, “Latency-aware VNF chain deployment with efficient resource reuse at network edge”, in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, IEEE, 2020, pp. 267–276.

[5] R. Gouareb, V. Friderikos, and A.-H. Aghvami, “Virtual network functions routing and placement for edge cloud latency minimization”, *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2346–2357, 2018.

[6] S. Khebbache, M. Hadji, and D. Zeghlache, “Scalable and cost-efficient algorithms for VNF chaining and placement problem”, in *2017 20th conference on innovations in clouds, internet and networks (ICIN)*, IEEE, 2017, pp. 92–99.

[7] Q. Zhang, F. Liu, and C. Zeng, “Adaptive interference-aware VNF placement for service-customized 5G network slices”, in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 2449–2457.

[8] 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.

[9] Z. Xu, Z. Zhang, W. Liang, Q. Xia, O. Rana, and G. Wu, “Qos-aware VNF placement and service chaining for iot applications in multi-tier mobile edge networks”, *ACM Transactions on Sensor Networks (TOSN)*, vol. 16, no. 3, pp. 1–27, 2020.

[10] N. Bouten, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, and F. De Turck, “Semantically enhanced mapping algorithm for affinity-constrained service function chain requests”, *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 317–331, 2017.

[11] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, “Demystifying the performance interference of co-located virtual network functions”, in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 765–773.

[12] M. Taghavian, Y. Hadjadj-Aoul, G. Texier, N. Huin, and P. Bertin, “An approach to network service placement reconciling optimality and scalability”, *IEEE Transactions on Network and Service Management*, 2023.