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

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*—Unavoidable transition from rigid dedicated hardware devices towards flexible containerized network services, introduced by Network Function Virtualization (NFV), brings novel opportunities while several new challenges. Meeting the expectations of NFV in post-5G networks depends on the efficient placement of the services. The online placement of the network services, demanding strict end-to-end latency requirements over the edge networks, 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 different cost functions to maximize the service acceptance. Extensive evaluations has 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. SDN separates the control and data planes, for centralized and intelligent management of networks using software applications. And, NFV virtualizes the entire classes of network functions using cloud computing technologies, to realize and manage the network services.

In NFV, the service placement is a crucial step which involves allocating physical resources for network services with specific Service Level Agreement (SLA) constraints, requiring heterogeneous resources. In terms of the complexity, the placement problem is proved NP-Complete by reducing the Bin Packing problem to it [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 Core Network (CN), Content Delivery Network (CDN), Internet of Things (IoT).

The placement of services is studied following many objectives. These objectives are addressed as an optimization problem, subject to various constraints including the resources, Quality of Service (QoS) requirements, and other constraints specific to the objective or the use-case. Resource optimizations, QoS optimizations (e.g., latency, availability, reliability, security), energy consumption, Revenue to Cost (R2C), and Service Acceptance (SA) are among the well-studied objectives [2]. We are interested in maximizing the SA as our objective. SA is defined as the number of services that can be placed over the network subject to the resources and QoS requirements.

Two general categories which are envisaged 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 the 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]. Following our objective, in offline placement, although the complexity is not negligible, by a well-defined Integer Linear Programming (ILP) model, we can perform an optimization to find the maximum number of the 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 the future service requests. This can be done by optimizing a cost function which is consistent with our objective. The scalability is another important requirement which needs to be considered in an online placement solution.

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 that is 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 of the services must necessarily stay on the edge, where we are not generous in spending these resources.

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 (BaB), capable of achieving the optimal and the near-optimal results according to several the cost functions and the search strategies.
- We explore several cost functions and demonstrate inconsistency of bandwidth and/or latency optimization, and the benefits of fair placement, with our objective of maximizing the SA.

The *fairness* of our approach can be explained in two ways. On the one hand, 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 on which 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. Next, we describe in detail our BaB approach in Section IV. Then, in Section V we plunge into the evaluations to investigate the effectiveness of our solution. Finally, we conclude and mention the future directions in Section VI.

II. Related Works

Studying the placement algorithms in the literature begins by the introduction of NFV. It is a commonly encountered in various use-cases, considering different categories of the objectives.

A significant part of the studied placement algorithms involves the exact solutions, where the problem is formulated as an ILP or Mixed Integer Linear Programming (MILP) and is solved with solvers and optimizers.

In [3], the authors investigate the placement of multi-constrained services to maximise the number of placed service requests, first by proposing an ILP resolution, and then a heuristic. They consider bandwidth and latency for the links, and computing resources for the nodes on an edge-core star-based network topology. Although strict requirements of E2E latency of the service requests and the anti-affinity rules are not studied, but limited node resources was considered on the edge, and they demonstrated that a fair distribution 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, but they proposed an interesting Depth-First Search (DFS) based algorithm for placing the Virtual Network Functions (VNFs) and the Virtual Links (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 a 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 the optimality, they could suffer from scalability issues. Performing an exact resolution could require a long execution time for obtaining an optimal solution, which could make restrictions for using in large-scale networks. This limitation presents a significant obstacle in online placement scenarios with time constraints.

Heuristic approaches are often retained, when it comes to the scalability. Best-Fit or Decreasing First-Fit is one of the most well-known and efficient placement heuristic, which places the VNF by sorting the nodes and placing the VNFs over the node with the maximum amount of available resources iteratively. [6] explores heuristic algorithms (especially the Best-Fit algorithm), and the authors propose a multi-stage graph algorithm to find near-optimal solutions in a scalable manner. They evaluate the execution time, the acceptance ratio, and the average cost of the placement in terms of 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), making it intolerable for some QoS-sensitive 5G use cases (e.g., autonomous driving and 4K/8K HD video).

Although, designed to obtain a solution in a reasonable time frame, heuristics do not provide quality guarantees, and the risk of getting stuck in local optimums is extremely high.

Meta-heuristic and evolutionary algorithms are another direction addressing 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. Despite being powerful in escaping from local optimums, these algorithms suffer from unpredictability,

especially in terms of execution time, which is extremely important in online placement.

Advanced AI search mechanisms and recent developments in machine learning techniques, and particularly Deep Reinforcement Learning (DRL), have gained lot of attention due to their promising potential. 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.

Although, the problem of the placement has been studied for a while in several related areas, but to the best of our knowledge, the placement problem 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 in the literature. Generally, the proposed placement approaches try to optimize whether the cost of the resources, or the QoS requirements like bandwidth and latency. A network service requires node resources (*a.k.a.* computing resources) for deploying its VNFs, and link resources (*a.k.a.* network resources) for realising its VLs. Supposing that we can not share the already placed VNF, and we need to provide what is required as node resources for deploying the VNFs, 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 will show that this optimisation approach could have a devastating effect on the edge networks with limited node resources, and it needs to be avoided. We will see that it can lead to draining the resources at 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 which need to be satisfied to achieve maximum SA.

Affinity and anti-affinity rules in cloud-computing provide a mechanism for establishing a trade-off between performance and reliability of the realised service. Affinity asserts putting VNFs on the same network node for increasing inter-networking performance. While, anti-affinity is used to improve the reliability and the availability by preventing certain VNFs of the same service from sharing the same physical resources in order to reduce the impact of a single network node failure [10]. In this paper, 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), for the following reasons. First, the anti-affinity rules is effective in satisfying the availability requirements of the services that need to be scaled over separate network nodes (scaling horizontally) to ensure that a node failure would not violate the availability of the whole service [11]. 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 [12]. Last but not least, not considering this constraint results in a remarkable relaxation of the placement problem, so that the gain of using different placement approaches becomes marginal.

III. Model

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	Delay experienced on link $e \in E$
D_k	Delay requested for service $k \in K$
D_l	Delay requested between the two VNFs of $l \in L_k$

The placement problem consists in 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 SLAs (reflecting the QoS requirements). A VNF $f \in F_k$ requests a subset of resources (here we consider 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, as well as QoS metrics (here we consider 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. The notations are summarized in Table I.

A. Compact formulation

Variables The ILP contains two sets of binary variables: the first set, x, represents the routing decision, and the second set, y, the service placement decision. More precisely, the variable $x_e^{l,k} = 1$ if the virtual link l of service $k \in K$ is using the link $e \in E$; and 0, otherwise. The variable $y_v^{f,k} = 1$ if the VNF $f \in F_k$ of the service $k \in K$ is instantiated in node $v \in V_f$; and 0, otherwise. Moreover,

for each service $k \in K$, we force $y_v^{f,k}$ to 0 for any node v that doesn't belong to V_f.

Constraints We consider the CPU as a node resource, and the bandwidth as a link resource. In addition, in terms of the QoS metrics, end-to-end latency for the entire service. Note that, our approach does not impose any limitations for adding more resource types or more QoS metrics for nodes, links, or services. A VNF can demand a set of resources of different types (e.g., CPU, GPU, RAM, storage, FPGA, etc.), which can be provided by the network nodes. Consequently, we consider two sets of capacity constraint:

$$\sum_{k \in K} \sum_{l \in L_k} B_l x_e^{l,k} \leq B_e, \tag{1a}$$

which is the bandwidth capacity constraint ($e \in E$); and

$$\sum_{k \in K} \sum_{f \in F_k} C_f y_v^{f,k} \leq C_v, \tag{1b}$$

which is the CPU capacity constraints ($v \in V$).

First, we ensure that the VNFs of a service are only placed once throughout the network with:

$$\sum_{v \in V} y_v^{f,k} \leq 1 \qquad \forall k \in K, \forall f \in F_k. \tag{1c}$$

Moreover, VNFs of the same service $k \in K$ cannot be placed on the same network node $v \in V$, ensured with:

$$\sum_{f \in F_k} y_v^{f,k} \leq 1, \qquad \forall v \in V_f, \forall k \in K. \tag{1d}$$

For each service $k \in K$, we ensure the virtual link $l = (f_i, f_j) \in L_k$ flow conservation at each node $v \in V$ with:

$$\sum_{e \in \omega^+(v)} x_e^{l,k} - \sum_{e \in \omega^-(v)} x_e^{l,k} = y_v^{f_j,k} - y_v^{f_i,k} \tag{1e}$$

Finally, we bound the end-to-end service latency. We consider the sum of the latency required by VLs as the end-to-end service latency, and the sum of the latency provided by placed network paths with the following constraints:

$$\sum_{e \in E} D_e \sum_{l \in L_k} x_e^{l,k} \leq D^k \qquad \forall k \in K, \tag{1f}$$

$$\sum_{e \in E} D_e x_e^{l,k} \leq D_l \qquad \forall k \in K, \forall l \in L_k. \tag{1g}$$

Objective The objective function is defined as:

$$\max \sum_{k \in K} \sum_{v \in V} y_v^{f_0,k}, \tag{1h}$$

where f_0 is the "first" function of the service. We only need to count the number of "first" functions in the objective since the flow conservation constraints ensure that a solution contains no partial embedding.

Cuts An ILP solver's performance depends on the problem's relaxation (*i.e.,* the problem without integrality constraints). It evaluates the nodes of its Branch-and-Bound tree with the linear relaxation of the problem. If the linear relaxation solution at the current node is worse than the best integer solution found so far, it can prune the node. The linear relaxation quality (the gap between the integer and relaxation solution) significantly impacts the resolution time since a good relaxation will lead to a faster exploration of the tree. We can improve the lower bound by devising cuts. These constraints are not necessary for the correctness of the model as they are implicitly satisfied by the constraints of the base integer model, but they are usually violated in the relaxed version of the problem.

This formulation has a poor relaxation, however, we can improve it by exploiting the one VNF per node constraint. Since nodes can host only one VNF of the same service, we can define the minimum amount of bandwidth used by each service. For example, let's consider a daisy chain service graph with 3 VNFs (F1, F2, and F3), and 4 VLs connecting F1 to F2, F2 to F1, F2 to F3, and F3 to F2. If each VL would require 1 unit of bandwidth, at least, we require four units of network bandwidth. In mathematical form, we can express the minimum bandwidth usage as:

$$\sum_{k \in K} \sum_{v \in V} \sum_{l \in L_k} B^{l,k} y_v^{f_0,k} \leq \sum_{e \in E} B_e. \tag{1i}$$

B. Embedding Decomposition

The previous formulation lacks scalability. To solve larger instances, we can use decomposition methods. Multiple decompositions are available. We focus on the *embedding decomposition*, where each variable represents a possible embedding of the service onto the physical network. This decomposition is suitable when multiple services share the same configuration (same bandwidth, same number of CPUs, and same latency requirements) as they can share the same embedding and we can easily compute the resource allocation.

Because the number of embeddings is exponential, an embedding-based formulation would require an exponential number of variables (columns). However, a solution only contains a few of them. To generate only useful columns, we use the CG algorithm [13]. The algorithm solves large-scale linear programs via a back-and-forth resolution of a *master problem* and one or multiple *pricing problems*. Starting from a *reduced* master problem, the master problem feeds dual values to the pricing problems, and the pricing problems feed improving columns to the master problem. For each service k, we need to generate an embedding γ among all possible embeddings Γ_k for the service k.

1) Master problem: We only need the set of variables $z_{k\gamma} \in \mathbb{N}$ to indicate the number of services k allocated on the embedding γ. Each embedding γ is defined by the amount of bandwidth it uses on each link e, denoted by $\delta_e(\gamma)$, and the number of CPUs used on each node v, denoted by $\theta_v(\gamma)$.

We ensure that, for each service $k \in K$, we do not embed more services than requested with:

$$\sum_{\gamma \in \Gamma_k} z_{k\gamma} \leq n_k, \tag{2a}$$

where n_k is the number of requests requiring the same service k. For each link $e \in E$, we define its capacity constraints as:

$$\sum_{k \in K} \sum_{\gamma \in \Gamma_k} \delta_e(\gamma) z_{k\gamma} \leq B_e, \tag{2b}$$

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

$$\sum_{k \in K} \sum_{\gamma \in \Gamma_k} \theta_v(\gamma) z_{k\gamma} \leq C_v. \tag{2c}$$

Finally, the objective function is written as:

$$\max \sum_{k \in K} \sum_{\gamma \in \Gamma_k} z_{k\gamma}. \tag{2d}$$

2) Pricing problems: Each service k has its own embedding sub-problem. The problem is similar to the compact formulation, so we reuse the same notations for the variables.

- $x_e^l \in \{0,1\}$ 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 $v \in V$.

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

$$\sum_{v \in V} y_v^f = 1. \tag{3a}$$

Unlike the compact formulation, we write these constraints as equalities to make sure that the service graph is embedded.

The second set of constraints ensures that each node $v \in V$ can only host up to one VNF of the service:

$$\sum_{f \in F_k} y_v^f \leq 1 \tag{3b}$$

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

$$\sum_{e \in \omega(v)} x_e^l - \sum_{e \in \omega^-(v)} x_e^l + y_v^{f_i} - y_v^{f_j} = 0. \tag{3c}$$

The overall delay of the service is ensured with:

$$\sum_{e \in E} D_e \sum_{l \in L_k} x_e^l \leq D^k, \tag{3d}$$

and the delay between each virtual link $l \in L_k$ with:

$$\sum_{e \in E} D_e x_e^l \leq D_k \tag{3e}$$

3) Pricing objective function: The pricing problems feed improving columns to the master problem. Improving and non-improving columns differ in their reduced costs as the reduced cost of a variable indicates the improvement of the objective function if the variable enters the solution. The goal of the pricing problem is 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 obtain 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 follows:

$$\min \quad \sum_{k \in K} n_k \pi_k^{(2a)} + \sum_{e \in E} B_e \pi_e^{(2b)} + \sum_{v \in V} C_v \pi_v^{(2c)} \tag{4a}$$

$$s.t. \quad \pi^{(2a)} + \sum_{e \in E} \delta_e(\gamma) \pi_e^{(2b)} + \sum_{v \in V} \theta_v(\gamma) \pi_v^{(2c)} \geq 1 \quad \forall k \in K, \forall \gamma \in \Gamma_k \tag{4b}$$

Columns in the master problem become constraints in the dual problem and are also in exponential numbers. Similarly to the CG algorithm, the row generation algorithm starts from a reduced problem and searches for any violated constraints. Given a dual solution $\bar{\pi}$, the separation problem of the dual searches an embedding that violates constraints (4b), *i.e.,* any embedding γ such that

$$\bar{\pi}^{(2a)} + \sum_{e \in E} \delta_e(\gamma) \bar{\pi}_e^{(2b)} + \sum_{v \in V} \theta_v(\gamma) \bar{\pi}_v^{(2c)} < 1. \tag{5}$$

Going back to the embedding sub-problem for a given service k, defined by constraints (3), the objective function becomes

$$\min \sum_{e \in E} \bar{\pi}_e^{(2b)} \sum_{l \in L_k} B_l x_{el} + \sum_{v \in V} \sum_{f \in F_k} C_f \bar{\pi}_v^{(2c)} y_{vf}. \tag{6}$$

If the optimal value of this problem is strictly less than $1 - \pi_k^{(2a)}$, 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 ILP and CG. We can obtain the global optimum (*i.e.,* the maximum feasible number of placed services) with ILP and CG in an offline placement scenario. But, when it comes to the online placement, since we do not know all of the service requests in advance, we need to ensure that the placement of the current service request improves the possibility of accepting the future service requests. This can be done by optimizing a cost function which is consistent with our objective. For simplicity, we use *network* and *service* to refer to the topology graphs of the Substrate Network and the Service Request.

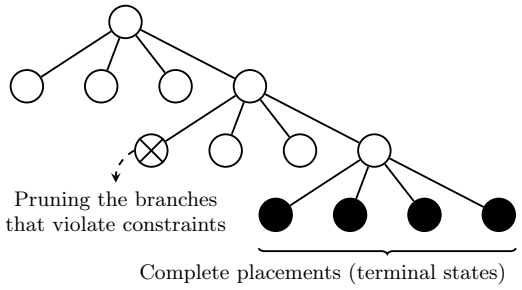


Figure 1. A sample BaB search tree, placing a service with 3 VNFs over a network with 4 nodes

BaB is one of the most popular algorithm design paradigms for solving optimization problems with exponential complexity. There are many types of constraints in the placement problem, which can be satisfied in BaB in an efficient way (the more constraints we have, the more we can prune the tree, resulting in faster search procedure). Moreover, BaB allows a complete and optimal search over the feasible solution space. Fig. 1 represents our BaB search tree. We refer to 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 contains a complete placement of a service over a network.

A. Expanding States

Starting at the root state of the tree, we select a VNF from the service and we generate the corresponding sub-states to each network node that can provide the required resources (a parent-child relationship). Each state contains a complete image of the network after placing the VNF over the corresponding node. Once a VNF is placed, we can place its outgoing and incoming VLs if both of their end-points are placed. In that case, we search for one of the shortest paths between both end-points to embed the VL, considering the required bandwidth and latency. After placing a VLs, the network image of the state is updated accordingly.

B. VNFs' Selection

At each state, we need to select a VNF to place it over the network. For this selection, we need an ordered list of the VNFs. In order to generate this list, we traverse the service in Breadth-First Search (BFS), starting from the entry point ("first" VNF) of the service (traversing in BFS allows us to have a partial placement of a connected sub-graph of the service on each state, making it possible to be evaluated against constraint violation).

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 evolves by continuing to select a VNF from the service and then placing it over the network nodes, generating their corresponding sub-states.

The root state of the search tree is located at depth zero, so the states of the first depth contain partial placements of the first VNF of the list. Consequently, the last VNF of the list is placed by the states of the last depth. Thus, if we are placing a service with N number of VNFs over a network with M number of nodes, the depth of the search tree would be N, and the maximum branching factor would be M. Note that our solution does not impose any constraints for sharing a previously placed network function between services. In case we want to share and use exclusively an already placed network function of the same type of the currently placing VNF (whether it is a PNF or a VNF). Since the already placed network function has a known network location (network node), the other branches corresponding to other network nodes are pruned (there are several works considering the VNF sharing, like [14]).

The branches are pruned because of a violation of constraints like:

- Not being able to provide enough node resources for the current VNF
- Not being able to find a path for the VLs connecting the currently placed and already placed VNFs because of the lack of the bandwidth
- Not being able to find a path that could satisfy the overall E2E latency requirement of the service
- Having placed the current VNF over a network node that we already placed another VNF of the same service over that node (anti-affinity)

C. Cost Functions

The cost function evaluates the states, to determine the next state for the expansion on each iteration of the search procedure. It is served as a measurement to allow comparing different states. Since the states represents different partial or complete placements of the service (they may have placed more or less VNFs and VLs), we need to assure that the proposed cost function can compare all of the states.

D. Search Procedure

The search procedure is an iterative procedure involving a list of the states (which we call it *fringe*, and it only contains the root state at the beginning of the search). On each iteration, we pop and expand the head state from the fringe, and we store the generated sub-states in the fringe. After expanding a state, we evaluate the sub-states against the constraints, and we only store the non-violating sub-states in the fringe (*i.e.,* they are pruned from the tree).

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 to 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 into

to the fringe. 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.,* UCS traversal is complete and optimal).

The search procedure continues the iterations until whether we succeed to find a terminal state, or we fail by reaching a timeout.

E. Search Strategies

We define a search strategy consisting of a cost function and a search traversal, that can be applied over our BaB formulation. We propose several search strategies, by following three approaches.

1) Latency Optimized Placement: We propose the LatUCS search strategy, which performs UCS traversal, over a cost function which is defined as the sum of the latencies of the realized paths that place VLs. The placements that LatUCS find are optimal (*i.e.,* they have the minimum feasible latency that can be realised for a service request). The idea is to investigate the effect of latency optimization on the objective of SA. It is worth mentioning that optimizing bandwidth instead of the latency would not change the results, since the bandwidth and the latency are optimized at the same time when the VNFs are placed as close as possible to the user-location and to each other.

2) Random Placement: We propose RanDFS search strategy, performing a DFS traversal, over a cost function that returns a random number. The idea is to investigate a worst-case random placement.

3) Fair Placement: We propose several search strategies to realize a fair distribution of the node resources. We propose 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.

In addition, we propose RecUCS and RecDFS, which perform UCS and DFS traversals over a cost function defined as the average of applying reciprocal function over the available resources for only the network nodes that place the VNFs of the service request ($cost = \frac{1}{r_i+1}$, the r_i represents the available resource of the network node which places the i-th VNF, and the $+1$ is added to prevent division by zero). We use the cost function proposed in [3], which tries to put a high cost for placing a VNF over a network node that is short on the resources.

V. Experimentation

Our implementations are made in Java, and we use the Gurobi solve our model. All of the evaluations are executed on a PC with an Intel Core i7-3687 CPU with 8GB of RAM.

To evaluate our search strategies following our objective of SA, we begin by initializing the network nodes and links with their full capacity of the resources. Then, iteratively, we generate a service request according to the specified parameters, then we try to place it over the network using the specified strategy. If we find a placement for the requested service, we apply the placement over the network by reserving its required resources, and we start a new iteration. Otherwise, we finish the evaluation and report the number of services successfully placed.

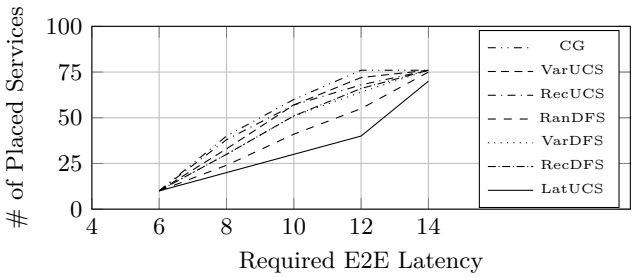


Figure 2. Number of placed services comprised of 3 VNFs requiring different E2E latencies

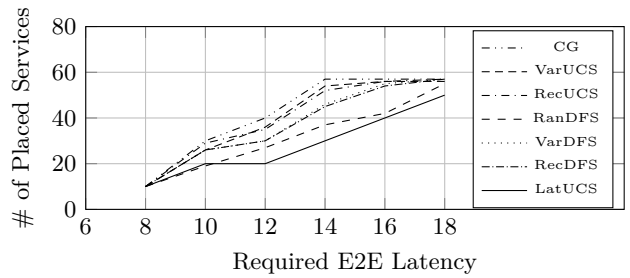


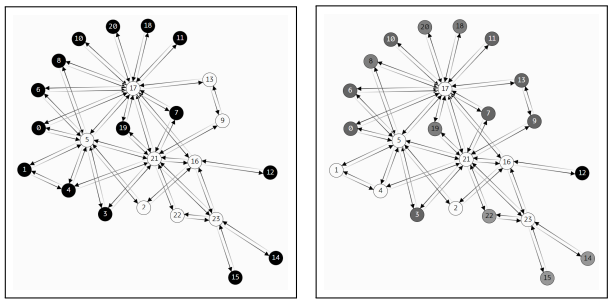
Figure 3. Number of placed services comprised of 4 VNFs requiring different E2E latencies

We perform our evaluations on the BT-Europe and BT-North-America network topologies (selected from the Zoo Topology dataset [15]), with respectively 24 and 36 nodes, and 74 and 152 bi-directional 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 3 to 5 VNFs with bi-directional VLs, and each VNF requires a single unit of CPU.

The user location is fixed at the network node *12* for BT-Europe, and the node *34* for BT-North-America. 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 Range

Latency is a crucial constraint, which 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-enough E2E latency, so that we can place it almost wherever we want, 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.



(a) (b)

Figure 4. Remaining available resources at the end of the evaluation by using LatUCS in (a), and VarUCS (b)

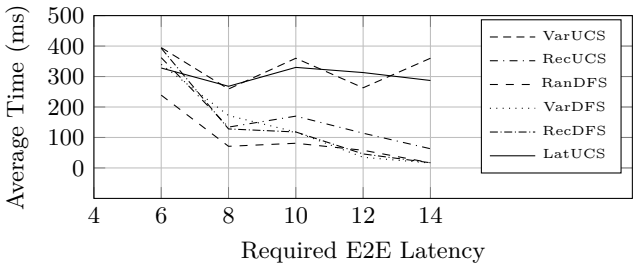


Figure 5. Average time of placing services comprised of 3 VNFs requiring different E2E latencies

B. Latency Optimization

Fig. 2 and Fig. 3 represent the number of placed services comprised of 3 and 4 VNFs requiring different E2E latencies over BT-Europe network. As it is shown in Fig. 2, all of the search strategies place almost the same number of services for the lowest feasible latency (6 units), and the high-enough latency (14 units). Considering the E2E latency of 8 to 12 (which we call it *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) Graph Representation of Latency Optimization Effect: Fig. 4 demonstrates remaining available resources of the network nodes at the end of the evaluation for BT-Europe network, placing serivces of 3 VNFs requiring 10 units of latency. We could make 30 placements using LatUCS (Fig. 4(a)), and 50 placements using VarUCS (Fig. 4(b)). In this figure the network nodes with full capacity are shown in black, the ones with no resources in white, and the others in gray-scale proportionate to their percentage of available resources. As mentioned before, by optimizing latency in LatUCS, we drain the resources around the user-location (fixed on node 12), making it impossible to place more than 30 services. By optimizing the variance of the available resources in VarUCS, we prioritize using the nodes having plenty of resources and we avoid using the nodes being short in the resources. Thus, we can reach out the resources on the nodes that are more distant, leading to place more services.

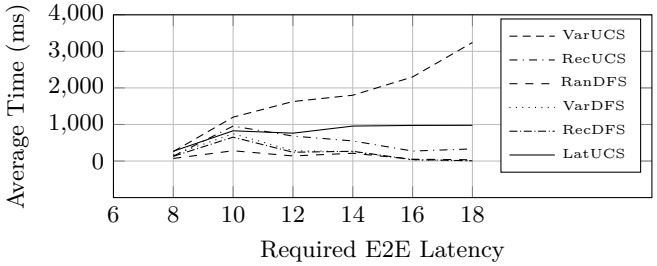


Figure 6. Average time of placing services comprised of 4 VNFs requiring different E2E latencies

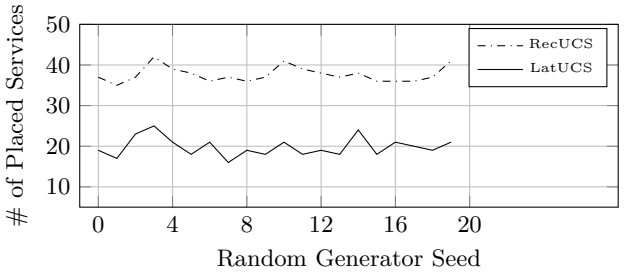


Figure 7. Number of placed services in each random experiment

2) Random Service Requests: Until now, we considered a fixed size for the service requests during an evaluation. But, in reality, the service requests can have different sizes and different latency requirements. Now, we generate random services, and repeat these evaluations many times with different seeds of randomness. In each random experiment, we start with generating a service with a random size in a range of [3, 5], with a random latency in the *effective* range based on its size, and continue placing them, until the specified strategy fails. We compare RecUCS, which performs a fair placement, with LatUCS, which perform latency optimization. As it is shown in Fig. 7 on average we can place 1.9 times more services using RecUCS instead of LatUCS.

C. 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. 5 and Fig. 6 show the average execution time of placing the services comprised of 3 and 4 VNFs requiring different E2E latencies over BT-Europe network. As it is shown in Fig. 5, RecUCS can place the services 1.95 times faster on average than VarUCS.

D. Big Picture

1) Service Acceptance: Extensive evaluations were performed, considering both networks of BT-Europe and BT-North-America, with service requests of different sizes (3 to 5 VNFs), requiring an *effective* range of E2E latency,

Table II
Our online strategies against global optimal(%)

RecUCS	VarUCS	RecDFS	VarDFS	RanDFS	LatUCS
93.1	90.7	82.2	81.4	66.9	50.8

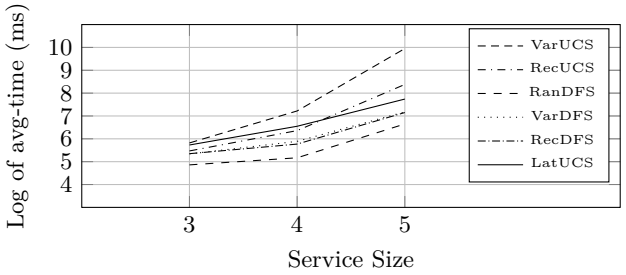


Figure 8. Logarithm of the average placement time for placing different service sizes on BT-Europe network

and user-location fixed on different network nodes. Table II shows the average percentage of the number of placements made by our proposed online placement strategies in comparison to our CG results which gives the maximum number of placements in an offline placement scenario.

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. 8. The complexity of our approach is clearly not polynomial (note that our problem is NP-Hard). The complexity of a 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

Although the VNF placement problem has been studied for many years, it still hot topic because of the constantly developing use-cases with new requirements and constraints. In this paper, we studied the online placement of the services on the edge networks with related constraints, 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] H. Zhu and C. Huang, "Availability-aware mobile edge application placement in 5G networks", in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[12] 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.

[13] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column generation*. Springer Science & Business Media, 2006.

[14] Y. Yue, B. Cheng, M. Wang, B. Li, X. Liu, and J. Chen, "Throughput optimization and delay guarantee VNF placement for mapping SFC requests

in nfv-enabled networks", *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4247–4262, 2021.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo", *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.