An Approach to Network Service Placement Reconciling Optimality and Scalability

Masoud Taghavian, Yassine Hadjadj-Aoul, Géraldine Texier, Nicolas Huin, Philippe Bertin

An Approach to Network Service Placement Reconciling Optimality and Scalability

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
‡Univ Rennes, Inria, CNRS, IRISA, France; firstname.lastname@irisa.fr
§Orange, France; firstname.lastname@orange.com

***Abstract*—The inevitable transition from physical dedicated hardware devices towards lightweight containerized reusable software modules with Network Function Virtualization (NFV) introduces countless opportunities while presenting several unprecedented challenges. Satisfying NFV expectations in post-5G networks heavily depends on the efficient placement of network services. In this paper, after modeling the placement problem and proposing the exact resolutions using Integer Linear Programming (ILP) and Column Generation (CG), we propose our deterministic placement solution, capable of obtaining optimal results with the scalability of a heuristic-grade approach. Our method is organized as a Branch and Bound (BnB) structure, applying Artificial Intelligence (AI) search strategies (especially A*) to address the problem of network service placement. We believe that it is suitable for a range of applications in online placement scenarios, whether we concentrate on the quality of the results or on the strict time constraints. We are interested in the popular objective of Service Acceptance (SA) maximization and have carried out several extensive evaluations. The obtained results confirm the effectiveness of our solution.**

***Index Terms*—Network Function Virtualization, Network Service Placement, Branch and Bound, Artificial Intelligence.**

I. Introduction

LEADING network operators adopted Software-Defined Networking (SDN) and Network Function Virtualisation (NFV) as a solution to ultra-flexible networks, meeting the diverse expectations of post-5G networks. Decoupling the data and the control plane in SDN and using cloud computing technologies to virtualize, orchestrate, and scale the entire class of network functions to realize services in NFV are indeed inevitable in modern networks [1].

Service placement represents one of the most important steps in NFV. It tackles the allocation of physical resources for network services, demanding heterogeneous resources with specific Service Level Agreement (SLA) constraints. In [2], the VNE placement problem is proved as a NP-Complete problem by reducing the Bin Packing problem to it. And it is encountered in many research areas and use-cases of post-5G networks, ranging from VNF Forwarding-Graph (VNFFG) placement and Network Slicing, to virtualization of the Core Network (CN), Content Delivery Network (CDN), Internet of Things (IoT), and more [3].

The placement of services is carried out according to 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 that are specific to the objective and the context. Resource optimizations, QoS optimizations (e.g., latency, availability, reliability, security), energy consumption, Revenue to Cost (R2C), and SA are among the well-studied objectives [4]–[7]. We are particularly interested in SA. SA is defined as the maximum number of services that can be placed successfully over the network subject to the resources and QoS requirements.

In [8], we addressed the problem of the online placement of services by proposing the basics of our BnB-based approach, enabling us to apply different AI search strategies. We demonstrated how to obtain optimal results (the distinguishing advantage of ILP-based placement methods) while maintaining the scalability of a heuristic-grade placement strategy.

In this paper, we extend our previous work in several aspects. We realized that, along with obtaining optimal results to maximize service acceptance, network fragmentation is a critical issue hardly addressed in the literature. Trying to mitigate this problem, we modified our approach with the fair placement of network load to avoid fragmentation. The experiments not only confirm the effectiveness of the used tweaks on the network defragmentation but also show a significant improvement in terms of SA. In addition, we noticed that the completeness of our node-based BnB formulation was at stake. Accordingly, we proposed a mixture of node-based and link-based formulations as a solution. From the early stages, we were keen to compare our approach with the offline optimal placement to see to what extent our approach was close to the general optimal solution. Consequently, we defined the mathematical model of our problem as an ILP. Following our intuition, we tried to solve our model using CG. Although we can find the optimal placements significantly faster than the ILP, scalability remains the major challenge in addressing our problem following an exact resolution. Fig. 4 summarizes the proposed approaches.

This paper is organized as follows. First, we explore the

related works in Section II. We present our formulation of the problem by an ILP and CG approach in Section III. Next, we describe in detail our BnB 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 Work

Placement algorithms have been studied in the literature for several years after the introduction of NFV. Placement is a commonly encountered concept in NFV, with various use-cases. These use-cases consider different categories of the objectives, including service acceptance and revenue to cost, to energy efficiency, security, resiliency, and availability.

Offline placement involves finding the global optimum for placing a batch of services based on known service requests and their requirements (i.e., in terms of resources and QoS metrics). In online placements, we look for an optimal placement (in terms of resources) of a single service request as soon as it arrives. The search for the global optimum is not possible in online placement because we have no information about the requirements of future demands. While finding a local optimum may appear an easier problem to solve, its complexity is still very high. In addition to the optimality, the growing need for placing multiple services as quickly as possible over the big networks exacerbates the situation.

A significant part of the placement algorithms involves the exact resolutions, formulating the problem as a Mixed Integer Linear Program and trying to solve it via solvers and optimizers.

In [9], the authors aim to optimise the placement of multi-constrained services, by considering the bandwidth and the latency for the links, and different system resources for the nodes on an edge-core star-based network topology graph. Their objective is to maximise the number of placed service requests, first by proposing an ILP resolution, and then a heuristic. In an effort to ensure a fair distribution of the resources, they considered the cost of using a resource proportionate to the remaining capacity of the corresponding node or link (i.e., the cost of using a resource on an overloaded node/link is much more than an underloaded node/link). The researchers then demonstrated that this slight modification results in a more fair distribution of resources and leads to an increase in the number of accepted services.

Despite the optimality and ease of use of ILP-based methods, they suffer from scalability issues. Note that performing an exact resolution requires a long processing time until obtaining an optimal solution, and it can only be used for small network topologies. This limitation presents a significant barrier in online placement scenarios with real-time constraints.

When it comes to the scalability of approaches, heuristics are often retained. Best-Fit or Decreasing First-Fit is one of the most well-known and efficient placement heuristic, which sorts the nodes and places the VNF over the node containing the maximum amount of available resources iteratively. [10] explores scalable and cost-efficient heuristic algorithms, especially the Best-Fit algorithm, and it proposes a multi-stage graph algorithm to find near-optimal solutions in a scalable manner. They assumed that service requests arrive randomly, considering a random life span for each service. The services are comprised of 3 VNFs, requiring random units of resources for VNFs and VLs. The simulation duration is fixed to a time unit for each run, investigating the execution time (including the scalability of the Best-Fit and the Multi-Stage algorithms), acceptance ratio, and average cost of the placement in terms of the assigned resources.

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 [11], a genetic algorithm based approach is proposed. They compared their results against the Best-Fit algorithm as a baseline algorithm, and they achieved higher request acceptance rates, stemming from more efficient resource utilization compared to a baseline greedy algorithm with a similar optimization objective.

Despite being powerful in escaping from local optimums, they 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 lots of attention due to their promising potential.

The authors in [12] explore the potential of DRL for maximizing the number of accepted services, satisfying the QoS requirements. They show that their DRL approach can learn the non-linear relation between QoS metrics and traffic, in order to be able to find better placements, resulting in more accepted services. [13] tries DRL with Relational Graph Convolutional Neural Networks (RGCN) for maximizing the service acceptance. In [14], a DRL approach utilizing Double Deep Q Networks is proposed to reduce the rejection ratio of the service requests, while improving the system's scalability through an offline training. A Q-Learning approach is investigated in [15], trying to establish a compromise between scalability and optimality in order to maximize the benefit of the service provider. They could obtain near-optimal solutions in a relatively short time, while considering the network load balance.

Our work falls into the latter category, discovering the potential of AI search strategies, in particular A^*. It is worth specifying the position of our work among ML-based approaches. In our contribution, we are able to find optimal (in the worst case near-optimal) placements, while aiming to be as scalable as possible. Whereas in ML-based approaches, the methods are remarkably scalable (after the training phase, the DRL-based approaches and the ML-

based algorithms can generate results in one step), while the target is to obtain the placements as close as possible to the optimal and near-optimal results by exploiting novel learning approaches. Furthermore, the approach we propose is explainable, unlike neural network approaches which are not. These are important differences between our work and the recent ML-based approaches.

III. Model

In this section, we define our placement problem model, and then we try to resolve this model by ILP and CG. The obtained results are presented later on the experimentation section.

A. The problem definition

The service placement problem consists in placing a service in the form of a graph on a substrate network, which is also described by a graph. Thus, a service graph $k \in K$, where K is the set of services, is denoted by a directed graph $H_k = (F_k, L_k)$, composed of a set of Virtual Network Functions (VNFs) F_k, connected via a set of Virtual Links (VLs) L_k, accompanied with SLA, reflecting the QoS requirements. A VNF $f \in F_k$ requires a subset of resources, here we consider CPU C_f, and each VL $l \in L_k$ connecting two VNFs requires an amount of bandwidth B_l. Similarly, a Substrate Network (SN) is a physical network represented by a directed graph $G(V, E)$ with its corresponding nodes V and links E. A node $v \in V$ has a capacity of resources, here we consider CPU denoted by C_v, and a link $e \in E$ has a bandwidth capacity denoted by B_e, as well as QoS metrics (here we consider latency).

The placement problem deals with finding mappings of a service's VNFs and VLs into network nodes and network paths, respectively. The notations of the model are summarized in Table I.

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 $l \in L_k$

B. 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 with $e \in E$; and 0, otherwise. The variables $y_v^{f,k} = 1$ if the VNF $f \in F_k$ of the service $k \in K$ is instantiated in node $v \in V$; and 0, otherwise.

Constraints We consider the CPU as a node resource, and the bandwidth as a link resource. In addition, in terms of the QoS metrics, latency is considered for the links and 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 several network nodes. Consequently, we consider two sets of capacity constraints:

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

which is the bandwidth capacity constraints of each link $e \in E$; 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 constraint of each node $v \in V$.

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

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

Moreover, VNFs of the same service $k \in K$ cannot be deployed on the same network node $v \in V$, which is ensured by:

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

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. For the sake of simplicity, 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:

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

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

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:

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

C. Embedding Decomposition

The previous formulation lacks scalability. To solve larger instances, we need to use decomposition methods. These methods are popular tools to make scalable ILP-based algorithms and they also provide higher bounds to evaluate the solutions.

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 shines 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 [16]. 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 the following constraints:

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

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 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 the constraint

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

The overall delay of the service is ensured with the constraint:

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

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

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

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:

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

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

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), or in other words, any embedding γ such that

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

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

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

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

So far, we tried to address the placement problem using ILP and CG. The evaluations in the experiment section show that, although CG performs pretty well on small graphs of the services and the networks, it is not scalable for bigger graphs. This is where our solution plays its role. For simplicity, we use *network* and *service* to refer to the topology graphs of the Substrate Network and the Service Request.

A. BnB-Based Service Placement

BnB is one of the most popular paradigms of algorithm design used for solving combinatorial optimization problems with exponential complexity.

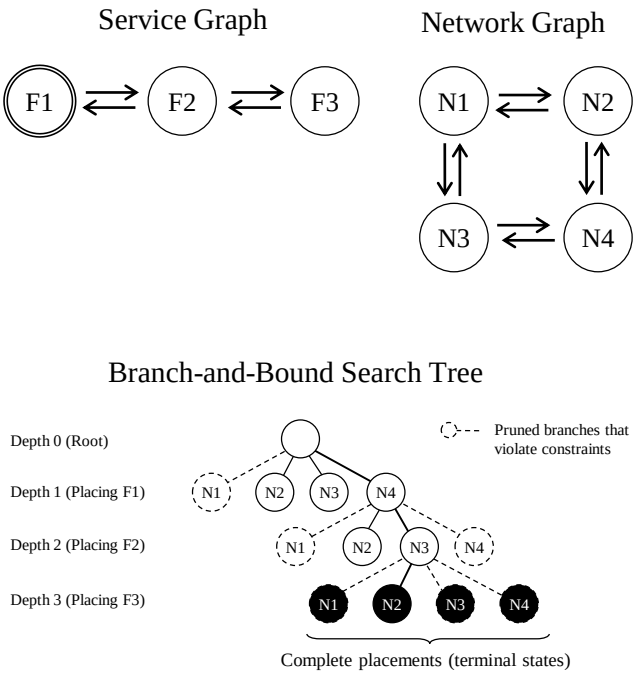


Figure 1. A sample BnB search tree, placing a service with 3 VNFs over a network with 4 nodes

BnB allows a complete search over the feasible solution space and avoids progressing in unfeasible branches by pruning the tree. We propose a BnB approach over a search tree (Fig. 1). To differentiate it from a network node, we designate a node of the search tree as a *state*. Each state carries placement information, as well as a complete image of the network (remaining resources). If a state gives the complete placement, we say that it is terminal.

1) Generating Search States: Starting at the root state of the tree, we select a VNF from the service and we generate a corresponding sub-state (a parent-child relationship) for each network node that can provide the required resources. Each state contains a complete image of the network, i.e., the remaining resources 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.

2) VNFs' selection order: 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. To decide which VNF to be selected in each step, we use the depth of the corresponding state and an ordered list of the VNFs of the service. In order to generate the list of VNFs, 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 violations. 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 [17]).

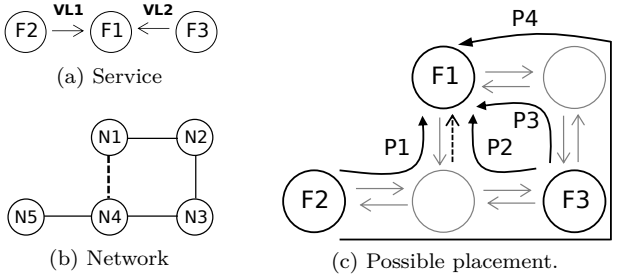
As it is shown in this figure, the branches are pruned because of a violation of constraints like:

- 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 (VNE constraint)

3) Search Strategy: After expanding the root state (generating all of its sub-states), we continue the search by selecting and expanding the states until we succeed to find a terminal state, or we fail by reaching a timeout. When we expand a state, we evaluate the sub-states against the constraints, and then we store them in an ordered list called the fringe (we do not add the violating states to the fringe, i.e., they are pruned from the tree). The next state to be expanded is chosen from the head of the fringe. The sequence of the states kept in the fringe specifies the direction of the search and it is determined by a search strategy. A search strategy specifies a traversal over the search tree (the order of visiting the states) through the fringe. The fringe can either be a queue (e.g., in BFS), a stack (e.g., in Depth-First Search (DFS)), or an ordered list (e.g., in A*). Our ultimate goal is to investigate the influence of applying various search strategies on the objective of SA (i.e., the maximum number of service requests that can be accepted). Accordingly, we will propose several search strategies.

B. Node-Based vs. Link-Based BnB

Until now, we tried to explain the structure of our BnB by focusing on a node-based placement approach. The states represented the possible placements for the VNFs over the nodes, while the VLs were placed in a known order. Since all of the network nodes are considered candidates for placing each VNF, we might think that the order of placing VNFs does not violate the completeness of our BnB (completeness guarantees to return a solution if at least any solution exists). But fixing the order of placing VLs and not considering the other possible sequences might violate the completeness. That is to say, it is possible to fail to find any placement (when all of the branches are pruned and we can not reach a terminal state), while we could have found a placement only by considering another order of placing VLs. Although this occurs only when the resources are scarce, it might happen.



(a) Service (b) Network (c) Possible placement.

Figure 2. The need for the link-based placement approach

The mentioned problem is illustrated in Fig. 2, in which we try to place a service (Fig. 2a) over a network (Fig. 2b). As shown in Fig. 2c, we have placed F1, F2, and F3 over N1, N5, and N3, respectively. Suppose that we decided to place VL1 before VL2. We have two options to place VL1 (i.e., over the path of P1 or P4.). We will choose P1 since it is the shortest path. Suppose that, by placing VL1 over P1, we use up the entire available bandwidth of the network link connecting N4 to N1. To place VL2, since the network link connecting N4 to N1 is taken by VL1, we cannot place VL2 on P2, but it is possible on P3 since it is available. Now suppose we decide to place VL2 first, and we decide to place it on P2 (both P2 and P3 are shortest-path candidates). Accordingly, if VL2 uses up the entire available bandwidth of the network link connecting N4 to N1, we will have to place VL1 over P4, which uses significantly more bandwidth and may violate latency requirements, leading to placement failure. By adopting a similar approach and redefining the BnB structure on links instead of nodes, we can consider all possible sequences of VLs placement.

C. A^* Based Search Strategies

1) A Bandwidth Optimized (ABO):* In ABO, we use A^* algorithm by concentrating on bandwidth usage optimization. A^* is amongst the most popular AI informed search algorithms due to its completeness, optimality, and optimal efficiency. It is considered the best solution for many problems in computer science. A* traverses the search tree according to f-costs, which for an arbitrary state of n is calculated by the sum of a cost function and a heuristic ($f(n) = g(n) + h(n)$). The states are stored in a list called fringe, an ordered list based on the f-costs. f is an estimation of the cost of an optimal solution from the root to a target state, the heuristic function h estimates the cost of an optimal solution from the current state to a target state, and g is the real cost of the path from the root to the current state. The optimality and completeness

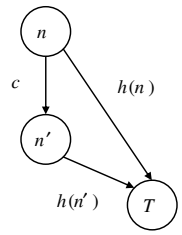


Figure 3. Illustration of the triangle inequality

of A* are proved if we choose an admissible and consistent heuristic function [18].

A heuristic that estimates the cost of the placement can be defined as the cost of placing each unplaced VL over a path containing exactly a single network link (since we need at least one network link to accommodate each VL). So, we consider the total amount of required bandwidth by all of the unplaced VLs as h, and g represents the total amount of bandwidth used for all of the placed VLs.

Theorem 1. *Our heuristic function is admissible.*

Proof. A heuristic function is admissible if it never overestimates the cost to the target state, as demonstrated in [18]. If we can place the source and destination of a VL over two adjacent network nodes, the cost of placement will be equal to what our heuristic estimates. Otherwise, we need a path of several links to accommodate a VL, which means the cost would be more than what our heuristic estimates. Thus, our heuristic never overestimates the cost to the target state, so our heuristic function is admissible. □

Theorem 2. *Our heuristic function is consistent.*

Proof. A heuristic function is consistent if the f-cost never decreases along a path from the root state to any arbitrary state as demonstrated in [18]. If n' is a successor state of n and c is the cost of placing unplaced VLs on n that are already placed on n' ($c = g(n') - g(n)$), we need to prove the triangle inequality of $h(n) \leq c + h(n')$, as illustrated in Fig. 3. We label the estimated cost of transition from n to n' as c' ($h(n') = h(n) - c'$). Knowing that the estimated cost of placement on n' is less than the estimated cost on n (since we have fewer unplaced VLs on n'), we just need to prove that $c' \leq c$. Since our heuristic does not overestimate the cost, this inequality is always true, i.e., our heuristic is also consistent. □

By proving Theorem 1 and Theorem 2, we can tell that our A^* search is complete and optimal. Note that, the optimality guarantees that if A^* succeeds to find a placement, no other placement exists using less bandwidth than the found one.

2) Fair A Bandwidth Optimized (FABO):* Along with the optimality, increasing the chances of accepting new service requests is also important to consider. Network fragmentation is one of the critical obstacles encountered during the placement. It can happen when the placement algorithm tends to place new service requests over the same nodes and links as long as they can provide the required resources, resulting in too many disconnected sub-networks. It is even more likely to happen if the algorithm is optimal and deterministic.

Fortunately, network fragmentation can be reduced by taking into account the relevant considerations in the placement algorithm. FABO is another search strategy that tries to address network fragmentation. On the one hand, it tries to avoid network fragmentation by sorting the states of the fringe according to the number of disconnected sub-networks (having at least two network nodes). On the other hand, it tries to make a fair distribution of the load over the network links by sorting the states of the fringe based on the variance of the remaining bandwidth of the network links. In a nutshell, the fringe is sorted with multiple criteria. The first criterion is the f-costs since FABO is defined on top of the optimal ABO. The second is the number of disconnected sub-networks, and the last one is the variance of the remaining bandwidth.

D. DFS-Based Search Strategy

Search strategies, regarding their types of traversal, are generally categorized into two classes: BFS-based and DFS-based. Note that BFS-based strategies include all of the traversals along with the breadth or a cost function, regardless of being informed or uninformed (therefore, ABO belongs to this category). DFS-based strategies traverse along with the depth of the search tree in order to quickly reach the terminal states. DFS Bandwidth Optimized (DBO) is another strategy that allows us to obtain high-quality near-optimal placements as fast as possible. In DBO, the fringe is a stack. The search tree evolves by popping a state from the fringe, expanding the state, sorting the sub-states by the amount of used bandwidth (g-cost), and then pushing back the sub-states into the fringe, until reaching a terminal state.

E. Best-Fit-Based Search Strategies

Best-Fit (BF) is one of the most well-known heuristic algorithms for service placement. BF sorts the nodes and places the VNF over the node containing the maximum amount of available resources. The placement fails to place a service as soon as it could not find any node to provide the required resources of a VNF of the service.

Looking for a way to see to what extent we can improve and benefit from the BF, we imagined the possibility to revise the previously placed VNFs. Consequently, we realized that our BnB structure is a perfect fit to put our idea into practice. By tweaking the DBO to sort the generated sub-states according to their amount of available node resources, instead of the used bandwidth, we can obtain another proposed search strategy called Enhanced Best-Fit (EBF).

F. Parallel Integrated Search Strategy

To maximize service acceptance, it is inevitable to minimize service rejection. Finding optimal placements and

preserving the resources as much as possible is necessary, but being unable to find an optimal placement within the time constraint, leading to service rejection, is not coherent with our objective. Therefore, it is essential to have backup strategies to find near-optimal placements as fast as possible. To escape from service rejection, we need to try multiple strategies. Parallel execution of the strategies allows each strategy an equal opportunity over time. Parallel Integrated (PI) is another search strategy that does not define a specific traversal, but it combines other strategies in order to obtain the best results in terms of service acceptance. In PI, we execute FABO, ABO, and DBO in parallel and wait until they finish (whether they succeed or reach the timeout). If multiple strategies succeed to find the placements, the one with the best quality is selected, i.e., the placement which is found by FABO is preferred over the one found by ABO, and similarly, ABO's result wins over DBO's.

V. Experimentations

A. Parameters and Steps of Evaluation

The evaluations are executed on a system with an Intel Core i7-3687 CPU and 8GB of RAM. Our implementations are made in Java and run on Windows 10. We set a timeout of 2s for placing a service in all of our evaluations (*i.e.*, if a strategy can not find a placement within the timeout, it fails).

We perform our evaluations over the BT-Asia-Pacific, BT-Europe, and BT-North-America network topologies, selected from the Zoo Topology dataset [19], with respectively 20, 24 and 36 nodes, and 62, 74 and 152 bi-directional links. We consider daisy chain, ring, and star topologies for service graphs having 3 to 10 VNFs and bi-directional VLs. Each VNF requires a single unit of CPU and a single unit of storage, and each VL requires from 1 to 10 bandwidth units. We use the SAMCRA [20] routing algorithm to place a VL over a network path while considering QoS metrics.

Our evaluation begins by initializing the network nodes/links with their maximum available resources. Then, iteratively, we generate service requests one by one based on the specified topology, size, and requirements. Then, we try to place them using the specified strategy within the timeout. On each iteration, if we find a placement for the requested service, we apply the placement over the network by reserving its required resources, and then we start a new iteration. Otherwise, we terminate the evaluation and report the number of services successfully placed.

B. Assumptions

To compare our strategies, we will introduce some assumptions that are not required for real scenarios but to highlight the behavior of the strategies.

1) No constraints for QoS metrics: The QoS metrics are systematically considered in the BnB tree construction, ensuring the absence of QoS violation.

2) No constraints for node resources: To illustrate the impact of node resource limitation on the placement strategies, we considered 10, 20, and 40 available units of CPU and storage resources for each network node (as our default configuration, we consider placing services with daisy chain topology, considering VNFs requiring one unit of CPU and storage for VNFs, and one unit of bandwidth for VLs, over BT-Europe network). Fig. 5 shows that all placement strategies act similarly when nodes have a small number of resources. The effectiveness of the strategies becomes clearer when network nodes have more resources. Therefore, from now on, unless mentioned otherwise, we consider an unlimited amount of CPU and storage for each network node and 10 units of bandwidth for each network link, at the beginning of the evaluation.

3) Placing VNFs of a service on separate network nodes: We respect the constraint of the Virtual Network Embedding (VNE) problem (*anti-affinity*), which does not allow placing any two VNFs of the same service over a single network node. 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 [21]. Moreover, not considering this constraint results in a remarkable relaxation of the placement problem, so that the different strategies can place almost the same number of placements. The anti-affinity is often partially considered in a service. Since, placing a single VNF over a single network node does not make a major difference in terms of the complexity in comparison to placing a group of VNFs (that need frequent communications) over a single node, here we consider anti-affinity for all of the VNFs of the service.

C. BF vs. EBF

Fig. 6 shows that we can place on average almost 3 times more services with EBF, compared to BF. We tried to place daisy chain services with different sizes over the BT-Europe network.

D. The Advantage of Integrating the Strategies

To evaluate the integration of strategies, we can imagine a strategy that executes DBO after ABO fails to find an optimal placement (we call it ADBO). Table II shows the number of placed services, and the percentage of the total remained network bandwidth at the end of the placement for ABO, DBO, and ADBO strategies. In general, ABO places more services and can save more remaining bandwidth. But, for services with 8 VNFs, although ABO saves a significant amount of bandwidth (43.24% vs. 7.03%), it places fewer services (30 vs. 38). Here, ABO fails to find an optimal placement within the time limit, and optimality is sacrificed for time, while, their combination (ADBO)

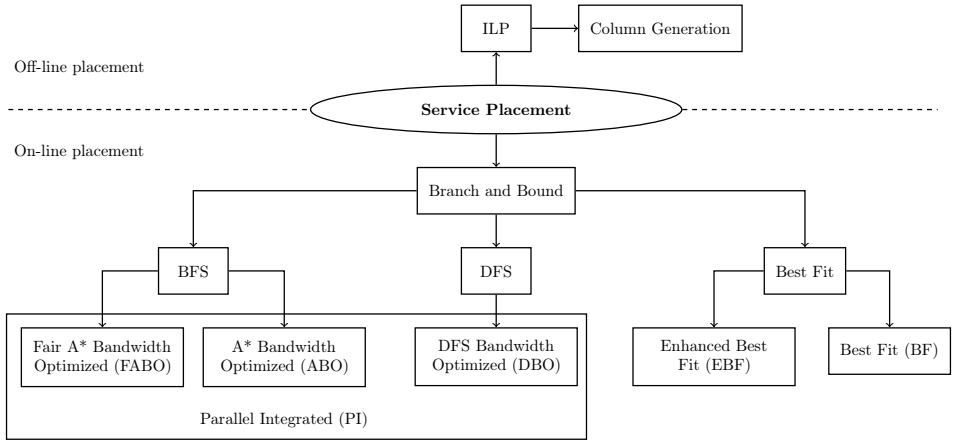
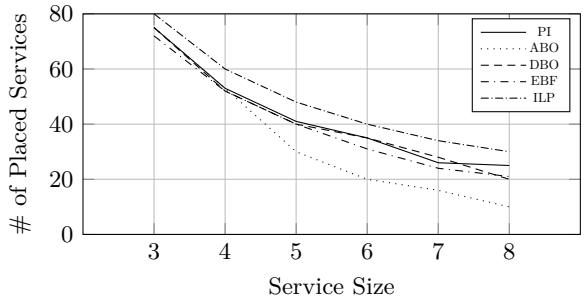
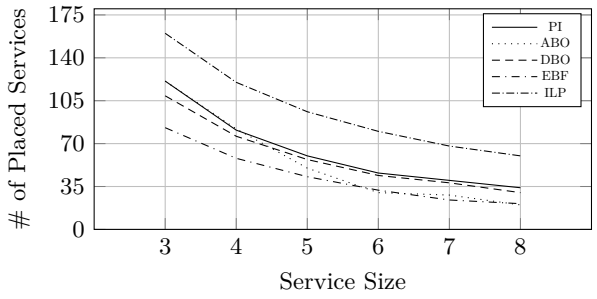


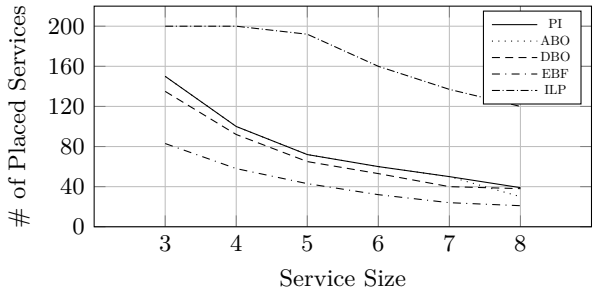
Figure 4. Overview of the proposition



(a) nodes with 10 units of maximum available resources



(b) nodes with 20 units of maximum available resources



(c) nodes with 40 units of maximum available resources

Figure 5. Illustration of node-resource limitation impact

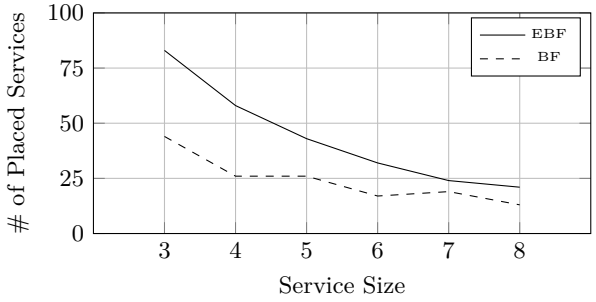


Figure 6. BF vs. EBF

Table II
The Advantage of Integrating the Strategies

Size	# of Placed Services			% of Remained BW		
	ABO	DBO	ADBO	ABO	DBO	ADBO
3	180	173	180	2.7	2.1	2.7
4	115	101	115	5.4	9.4	5.4
5	78	78	78	13.5	8.3	13.5
6	60	57	60	18.9	12.9	18.9
7	55	46	55	6.7	12.1	6.7
8	**30**	**38**	**39**	**43.2**	**7.0**	**17.5**

places more services and leaves the remaining bandwidth of 17.5%. The time restrictions on online placement argue for the integration of strategies to minimize service rejection. Accordingly, we introduced PI by parallel integration of FABO, ABO, and DBO.

E. Service Acceptance

Extensive evaluations were performed. Considering 10 different values of VL bandwidth requirement, 3 types of service topologies, 8 service graph sizes, and 6 strategies, we performed 4320 evaluations (1440 evaluations for each of the three network topologies). The results confirm that PI is almost always superior to the other strategies, making us able to measure them all against PI (Table III). We used quartiles (Q1, Q2 (Median), Q3) as well as min-avg-max, in order to represent the results. For example, an improvement of 494% on average means that PI places 5.94 times more services in comparison to BF considering

Table III
SERVICE ACCEPTANCE IMPROVEMENTS, GAINED BY USING PI INSTEAD OF OTHER STRATEGIES (%)

Strategy	Q1	Q2	Q3	Min	Average	Max
FABO	0	19	50	0	**115**	3800
ABO	0	15	33	-14	**26**	300
DBO	12	27	50	-14	**150**	4800
EBF	100	112	173	0	**208**	2400
BF	217	325	600	90	**494**	2400

Table IV
EVALUATION OF THE PI STRATEGY WITH THE ILP AND THE CG ALGORITHM

Size	ILP		CG & BnB		PI Strategy	
	#	Time	#	Time	#	Time
3	185	757	185	6	185	30
4	123	1857	123	22	121	29
5	92	12567	91	71	90	26
6	-	-	73	246	72	28
7	-	-	60	571	59	31
8	-	-	49	1113	47	29

the same evaluation parameters. Beyond these satisfactory results in terms of average, the max column shows that PI can place 25 to 49 times more services, while the placement problem becomes so complicated that DBO and EBF could not find any placement or found only 1 or 2.

F. The ILP and CG Results

We compare, in Table IV, the PI strategy with the offline solutions provided by the ILP and the CG algorithm. Both mathematical formulations were solved using the Gurobi solver. Solving the ILP becomes more difficult as the size of the services increases; it takes more than three hours to find the optimal solution for daisy chains of size 5. For larger instances, we rely on the CG algorithm to provide the solution and its upper bound. We see that our PI strategy finds close-to-optimal solutions, with an average gap of 2%.

Note that we can't expect an online placement solution like PI to reach the global optimum of the offline problem. It should also be noted that PI achieves this performance while keeping the computation time around 30s. Even though the CG algorithm shows better scalability, it still needs about 20 minutes to find a solution for daisy chains of size 8.

Furthermore, we were interested to see to what extent our placements found by the PI strategy are optimal. Note that the placements found by FABO and ABO are guaranteed to be optimal, but the ones found by DBO are not necessarily optimal. By performing a complete evaluation over the BT-Europe network, we witnessed that 89% of the placements were found by FABO and ABO, indicating that at least 89% of the placements are optimal.

Fig. 7 depicts a general comparison between different strategies except for PI. The behavior of the PI strategy depends on the timeout. In fact, by considering a very short period as a timeout, the PI strategy acts like DBO; by considering longer periods, it acts like ABO; and if we give it enough time, it acts like the FABO strategy.

Table V
EVALUATION OF THE OTHER STRATEGIES

Size	ABO		DBO		EBF		BF	
	#	Time	#	Time	#	Time	#	Time
3	185	1.8	173	0.9	82	0.4	44	0.2
4	106	1.8	102	0.7	58	0.4	26	0.2
5	82	1.4	78	0.6	43	0.4	26	0.3
6	61	3.8	58	0.5	32	0.5	17	0.2
7	52	3.5	46	0.5	25	0.4	19	0.2
8	31	3.6	37	0.5	20	0.4	13	0.2

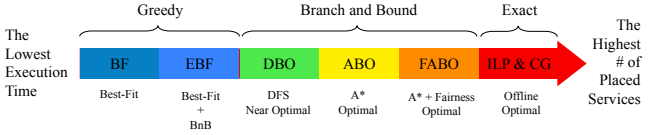


Figure 7. A general comparison between different strategies

The execution time of the other strategies are shown in Table V for more information.

G. Fair-Placement and Network Defragmentation

Up to this point, we have considered a fixed size for the service requests during an evaluation. Yet, in reality, the arriving service requests can contain different amount of VNFs and VLs with different resource requirements. We try to generate random services, and to guarantee the reliability and the repeatability of the results, we perform this scenario many times with different seeds of randomness (trying to perform Monte Carlo experiments [22]). Since the number of placed services (our random variable) is bounded (because our resources are limited), we can rely on the convergence in terms of the average of the outcomes of our random experiments. In each random experiment, we start with generating a daisy chain service with a random number of VNFs (with a uniform distribution) in a range of [3, 8], and continue placing them, until the specified strategy fails.

We want to compare the PI strategy, which performs a fair placement and avoids network fragmentation, with the ABO strategy, which only performs optimal placement. Fig. 8 represents a single random experiment over the BT-North-America network. The number of created sub-networks confirms the effectiveness of PI, not only in keeping the network's integrity until almost the end of the simulation but also in its capability to place more services than ABO.

If we take the sequence of the number of sub-networks on the experiment related to the Fig. 8 as we continue placing the services over the network, the average of this sequence will be 3.09 for ABO, and 1.06 for PI strategy. We performed 100 random experiments, with different seeds of randomness for each of the strategies and for all of our three networks. Fig. 9 shows the min-avg-max over the average number of sub-networks during each random experiment. The average obtained using ABO on BT-Asia, BT-Europe and BTN-America is 1.18, 1.38 and 1.82 (note that it increases according to the size of the network), while

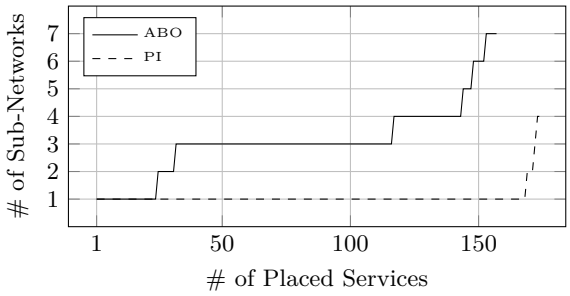


Figure 8. Network Fragmentation on BT-North-America

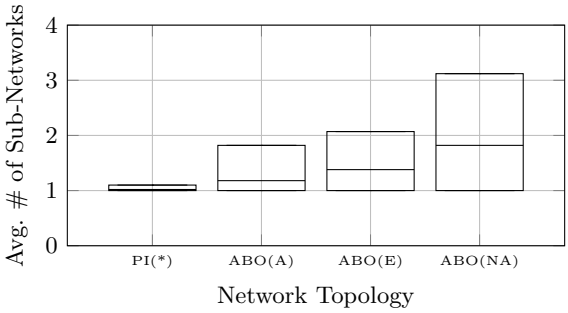


Figure 9. Network Fragmentation on different networks including BT-Asia, BT-Europe, and BT-North-America

it is 1.02 using PI over all of the results obtained for all these networks.

The number of placed services in 100 random experiments over the BT-Europe network is represented in Fig. 10. Fig. 11 demonstrates the cumulative average of the results of Fig. 10. The superiority of the PI strategy can be witnessed not only by the convergence of the cumulative average of PI towards 80.5 (95% confidential interval with 0.9% margin of error) and ABO to 68.4 (95% confidential interval with 1.7% margin of error) but also the fact that PI could almost always place more services than ABO considering each random experimentation.

H. Comparison with ML-Based Approaches

The authors in [12] propose a DRL-based approach for maximizing the SA, considering the QoS requirements. They compare their results with similar configurations as our approach (like using Bt-Europe topology and similar

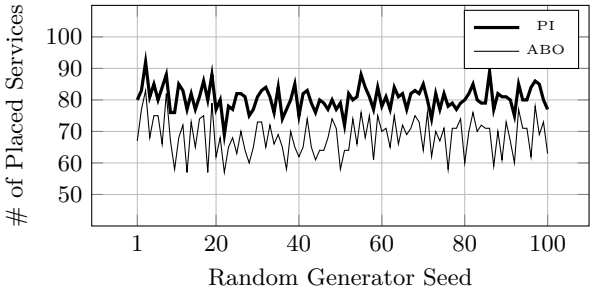


Figure 10. The number of placed services in each random experiment

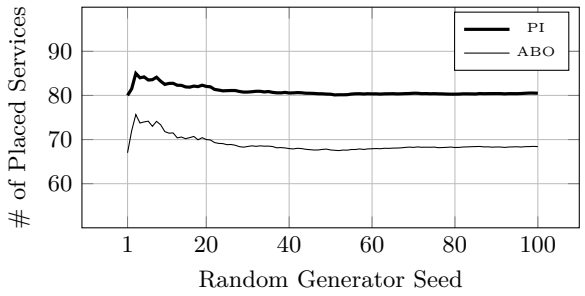


Figure 11. The cumulative average of the number of placed services in each random experiment

resources), with the First-Fit (FF) algorithm (FF is a simplified version of the BF algorithm, in a way that it does not sort the network nodes before placing each VNF). They defined the possible states, the actions and the rewards for the agent of the DRL to find the placements for the VNFs of a service. They used Deep Deterministic Policy Gradient (DDPG) in order to enhance the performance of DRL agent. They demonstrated almost 30% of improvements in the number of accepted services in comparison to the FF algorithm. [13] is another work proposing DRL and with RGCN for maximizing the service acceptance, and they showed almost 20% of improvements in the number of accepted services in comparison to the BF algorithm.

In ML-based approaches, we make the placements in two phases. At first, we train our model by the making random placements to let our agent learn to maximize the number of placements. Then, we evaluate our model to see how many placements it can make. Although the obtained results of the this approach are not as good as the results of our proposed approach. But, there are particular advantages by using DRL. In fact, after the training phase, the DRL-based approaches like all of the ML-based algorithms, we can generate results super fastly, making it ideal for large scale placement problems.

VI. Conclusions and future work

Although the VNF placement problem has been studied for many years, the need for an approach that could find a fair compromise between optimality and scalability still exists. Recent advances in AI and ML techniques have revolutionized many research domains. We proposed a solution based on a BnB structure, adapting AI search strategies, particularly A^*, capable of finding optimal placements considerably fast. Substantial empirical analysis has been made, and the results confirm a considerable improvement (494% on average) in comparison to the popular placement algorithm of Best-Fit.

As a downside of our A^*-based strategies, we could mention the exponential memory complexity of $O(b^d)$ (b stands for branching factor and d for depth). Yet, since the online placement imposes a time constraint, we only need to ensure that the search procedure will finish before using up the entire memory, depending on the scale of the network and the service. In addition, since there are

effective algorithms like Simplified Memory Bounded A^* (SMA*) to address it, we do not consider it a significant issue.

As a future direction, we are interested in the edge networks, where the location of the service consumers affects the service location. Besides, we are eager to see how to adapt our approach to the edge and to the hierarchical networks using, particularly, advanced ML techniques to boost our approach.

References

[1] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing", *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[2] 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.

[3] ETSI, *Network functions virtualisation (nfv) use cases*, https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports.

[4] 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.

[5] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining", *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, 2018.

[6] W. Yang and C. Fung, "A survey on security in network functions virtualization", in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, IEEE, 2016, pp. 15–19.

[7] M. Schöller, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions", in *2013 5Th international congress on ultra modern telecommunications and control systems and workshops (ICUMT)*, IEEE, 2013, pp. 208–214.

[8] M. Taghavian, Y. Hadjadj-Aoul, G. Texier, and P. Bertin, "An approach to network service placement using intelligent search strategies over branch-and-bound", in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[9] 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.

[10] 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.

[11] P. Rodis and P. Papadimitriou, "Intelligent network service embedding using genetic algorithms", in *2021 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2021, pp. 1–7.

[12] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding", *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[13] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: A novel deep reinforcement learning approach for beyond 5g network slicing", in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2021, pp. 1–6.

[14] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks", *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2019.

[15] J. Sun, G. Huang, G. Sun, H. Yu, A. K. Sangaiah, and V. Chang, "A q-learning-based approach for deploying dynamic service function chains", *Symmetry*, vol. 10, no. 11, p. 646, 2018.

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

[17] 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.

[18] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[19] 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.

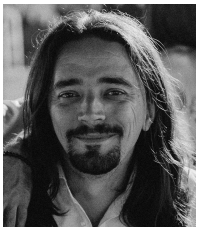
[20] P. Van Mieghem and F. A. Kuipers, "Concepts of exact qos routing algorithms", *IEEE/ACM Transactions on networking*, vol. 12, no. 5, pp. 851–864, 2004.

[21] 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.

[22] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications", *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

Masoud Taghavian received his B.Sc. in Information Technology from Tabriz University, Iran, in 2012, and a M.Sc. degree in Software Engineering in Shahid-Beheshti University, Iran, in 2015. He is currently pursuing a PhD at the Network and Connectivity laboratory at b<>com Institute for Research and Technology, Rennes, France, and IMT Atlantique, Rennes, France, working on the VNF placement in Network Function Virtualization using Artificial Intelligence and Machine Learning.

Géraldine Texier is a professor at the Network Systems, Cyber Security and Digital Law (SRCD) department at IMT Atlantique and a member of the Adopnet research team from IRISA, in Rennes, France. She is working in the Advanced Connectivity lab of the b<>com Institute of Research and Technology (IRT). She received her PhD degree on the management of the cooperation and the awareness in computer supported collaborative work in 2000 from the University of Rennes 1 (France). In 2016-2017, she was a visiting researcher at INRIA@Silicon Valley, CITRIS, University of California, Campus of Berkeley where she worked on routing for IoT in Smart Cities. Her main research topics are focused on network virtualization, routing and quality of service in communication networks.



Nicolas Huin is an associate professor at IMT Atlantique in the Network Systems, Cyber Security and Digital Law (SRCD) department. He received his PhD in 2017 from Université Côte d'Azur. His work focuses on the optimisation of Virtualized Networks.

Yassine Hadjadj-Aoul is currently working as full professor at the University of Rennes, France, since 2022, where he is also a member of the IRISA Laboratory and the INRIA project-team Ermine. He received a B.Sc. in computer engineering from Mohamed Boudiaf University, Oran, Algeria, in 1999. Dr. Hadjadj received his Master's and Ph.D. degrees in computer science from the University of Versailles, France, in 2002 and 2007, respectively. He was also a post-doctoral fellow at the University of Lille 1 and a research fellow, under the EUFP6 EIF Marie Curie Action, at the National University of Dublin (UCD). Then, he joined the University of Rennes as an associate professor. His main research interests concern the fields of wireless networking, congestion control, QoS provisioning, and Network virtualization.

Philippe Bertin is a Senior Research Engineer at Orange Innovation. He is also leading research with b<>com Institute for Research and Technology. He received his PhD degree in Computer Science in 2010 from the University of Rennes 1, and his Master degree in 1993 from the Universities of Versailles and Paris 6. His research topics are focused on cloud native 5G and beyond networks.