An Approach to Network Service Placement using Intelligent Search Strategies over Branch-and-Bound

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

An Approach to Network Service Placement using Intelligent Search Strategies over Branch-and-Bound

Masoud Taghavian*†, Yassine Hadjadj-Aoul*‡, Géraldine Texier *†, 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*—Network Function Virtualization (NFV) has been a significant shift from traditional dedicated hardware devices towards reusable software modules running over lightweight virtualized environments. While it brings many promising opportunities, it introduces several unprecedented complexities that need further considerations. Efficient placement of services is essential for achieving NFV expectations. We propose a highly reliable solution for systematically placing network services, touching the optimal results while maintaining the scalability, making it suitable for online scenarios with strict time constraints. We organized our solution as a Branch and Bound search structure, which leverages Artificial Intelligence (AI) search strategies (Especially A-Star) to address the placement problem, following the popular objective of Service Acceptance (SA). Extensive empirical analysis has been carried out and the results confirm a significant improvements.**

***Index Terms*—Network function virtualization, branch and bound, artificial intelligence, a-star.**

I. Introduction

The need for increasingly flexible networks with the ability to instantly integrate new services has led network operators to adopt Network Function Virtualisation (NFV) and Software-Defined Networking (SDN) in 5G networks. By separating the control and data planes, SDN enables networks to be managed centrally and intelligently, using software applications. Meanwhile, NFV uses the cloud computing technologies to virtualize the entire classes of network functions, connecting and chaining them together to realise services. Service Chain Description (SC-D), Service Chain Composition (SC-C), Virtual Network Function-Forwarding Graph Placement (VNF-P) and VNF Scheduling (VNF-S) are the four stages of Service Function Chaining (SFC) [8].

To satisfy the expectations of NFV, sophisticated algorithms are required to allocate the physical resources in an optimal way, within a fair time constraint. Resources are allocated by determining the suitable placements of the service requests over the Substrate Network (SN). The placement problem is classified as an NP-Hard problem in terms of complexity, i.e., it is a generalization of the Virtual Network Embedding (VNE) problem which has already been proven to be NP-Hard [2], [7].

A considerable part of the literature on the placement problem is dedicated to the exact resolution of the problem, which is generally formulated as an Integer Linear Programming (ILP) [1], [5]. Although they are powerful and optimal, they suffer from scalability and exponential time complexity. Using Meta-Heuristics is another approach to address the problem [4], [10]. In spite of being effective, they can suffer from convergence problems and unpredictability. Besides, they imply a search over the whole solution space, which is a waste of time considering the large number of constraints inherent to the problem (related to nodes, links, Quality of Service (QoS), different objectives, etc.), and it makes a huge proportion of the solutions infeasible. Meanwhile, using Artificial Intelligence (AI) and Machine Learning (ML) techniques for the placement problem is quite recent but highly promising. [6], [16] apply ML and [3], [9], [11], [14] propose solutions in Deep Reinforcement Learning (DRL) and Q-Learning for the placement problem. Most of the works in this category, until recently, use greedy heuristic search. Despite of being super fast, they are not optimal, and the risk of getting stuck in local optimums is extremely high.

In this paper, we address the placement problem, proposing a fast reliable solution for systematically placing network services based on a Branch and Bound (BB) method, which allows us to give different AI search strategies (especially A*) the opportunity to develop in this field of study. We will see how to touch theoretically proved optimal results (The distinguishing advantage of ILP-based methods), while maintaining the scalability of a heuristic-grade placement strategy. It is suitable for on-line scenarios that we need to conform to a time constraint. We adopted the popular objective of Service Acceptance (SA), which distinguishes the strategies based on the number of successfully placed services.

It is worth mentioning some of the motivations driving our approach. BB approach caught our attention for several reasons. First, we have a large number of different types of constraints in this problem, which are leveraged by BB in a beneficial way (the more constraints we have, the more we can prune the tree and limit the branches, leading to faster search procedure). Another advantage of

BB is the guarantee of not producing infeasible placements in any circumstance. It systematically respects all of the constraints as the search advances and avoids to progress in non-promising constraints-violating areas. It is suitable when we need to result a good-enough solution as fast as possible and to be able to improve it as we give it more time. That is why we think BB is a good choice for online placements. It also has an intrinsic potential of making use of parallel execution.

In the following, after formal specification of basic constraints, we investigate the source of the complexity, and propose our approach. We conclude with our meticulous evaluations to assure the effectiveness of our method.

II. The placement model

The placement consists in finding a pair of mappings of the VNFs and the Virtual Links (VLs) of a service into the nodes and the paths (sequence of links) of a SN. These mappings specify for each VNF the serving node, and for each VL data flow the sequences of links over which it is conducted. A Service Graph (SG) is denoted by a directed graph, composed of VNFs, connected via VLs, accompanied with a Service Level Agreement (SLA), reflecting the acceptable QoS requirements. The SN is the physical network represented by a directed graph with its corresponding topology, the available resources related to nodes and links, as well as QoS metrics associated with the links (*e.g.*, Latency, Loss, etc.). We are interested in online placement of services, so, we place a service as soon as it arrives [8]. We use Z-notation [13] to make a formal specification of the basic constraints of the placement. Table I introduces the sets and functions, utilised in the specifications of the constraints. Note that the primitive types like sets, relations, functions and sequences as well as several operators like *dom* and *ran* (domain and range of a function) are already specified mathematically in Z-notation. A VL is placed over a path which is a sequence of distinct links (1), chained together (2), in order to connect two nodes of the SN. In order to avoid loops, we add (3). We consider available amounts for different types of resources for a path, similar to a link, which is defined as the subsection or minimum of the available amounts for different types of resources of the paths links (4).

$$\begin{gathered}\forall p \in P \mid p = seq\ L \\ \forall i, j \in \mathbb{Z} \mid i, j < \#p \ \wedge\ i \neq j \Rightarrow p(i) \neq p(j)\end{gathered} \tag{1}$$

$$\begin{gathered}S(p(0)) = S(p) \quad \wedge \quad D(p(\#p - 1)) = D(p) \ \wedge \\ S(p(i)) = D(p(i-1))\end{gathered} \tag{2}$$

$$S(p) \neq D(p) \quad \wedge \quad S(p(i)) \neq S(p(j)) \tag{3}$$

$$A(p,t) = \bigcap_{i=1}^{n} A(p(i),t) \ \ ,\forall t \in \Delta \tag{4}$$

The available amount of resources of a node must be adequate to provide the amount of resources required by all the VNFs of a SG that have been placed on that node (5). Similarly, the available amount of resources of a link must satisfy the amount of resources needed by all of the paths placed on that link (6). The placement is complete, when all of the VNFs and VLs of the SG are placed, *i.e.,* the domain of the M and N functions are equal to the V and E sets respectively ($dom\ M = V \ \wedge \ dom\ N = E$).

Table I
Notations

Name	Description
V	The set of Virtual Network Function (VNF)s
N	The set of nodes
E	The set of Virtual Link (VL)s
L	The set of links
P	The set of paths (a path is a sequence of links)
∇	The set of node-resource types, e.g., CPU and Storage
Δ	The set of link-resource types, e.g., Bandwidth
$A(x,t)$	A function, representing an integer ($ran\ A \in \mathbb{PZ}$) indicating the available amount of a resource type (t) of a specific node, link or path (x). The first argument can be a node ($x \in N$) , a link ($x \in L$) or a path ($x \in P$). For the second argument, if the first argument is a node, it is node-resource type ($t \in \nabla$), otherwise, it is a link-resource type ($t \in \Delta$)
$R(x,t)$	A function similar to $A(x,t)$ but it indicating the requiring amount of specific resource type (t) of a specific VNF or VL (x).
$S(x)$ or $D(x)$	Source or destination of a VL ($x \in E$), Link ($x \in L$) or Path ($x \in P$). Source or destination of a VL represents a VNF ($ran\ S \in \mathbb{P}V$), otherwise it represents a node ($ran\ S \in \mathbb{P}N$)
$M(v)$	A function, representing the node on which a VNF is placed ($v \in V$ and $ran\ M \in \mathbb{P}N$)
$M^{-1}(n)$	A relation which represents the set of VNFs which are placed on a node ($n \in N$ and $ran\ M^{-1} \in \mathbb{PP}V$)
$N(v)$	A function, representing the path on which a VL is placed ($e \in E$ and $ran\ N \in \mathbb{P}P$)
$N^{-1}(l)$	A relation, representing the set of VLs which are placed over the paths that include a specific link ($l \in L$ and $ran\ N^{-1} \in \mathbb{PP}E$). To clarify $N^{-1}(l) = \{\forall e \in E,\ \forall p \in P \mid e \mapsto p \in N \wedge l \in ran\ p \bullet e\}$

$$\sum_{v\ \in M^{-1}(n)} R(v,t) \ \leq \ A(n,t) \quad ,\forall n \in N,\ \ \forall t \in \nabla \tag{5}$$

$$\sum_{e\ \in N^{-1}(l)} R(e,t) \ \leq \ A(l,t) \quad ,\forall l \in L,\ \ \forall t \in \Delta \tag{6}$$

III. Proposed solution

A. Branch-and-Bound-Based Service Graph Placement

1) Branch and Bound: BB is generally used for solving combinatorial optimization problems that are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. BB partitions the total set of feasible solutions into smaller subsets and systematically avoids non-promising branches of the tree by pruning. Our proposed BB approach is organized as a tree search (figure 1). Here, we use the the term state, to refer to the nodes of the search tree, to better differentiate the nodes of SN and search tree. Each state includes a status of the SN (capturing remaining

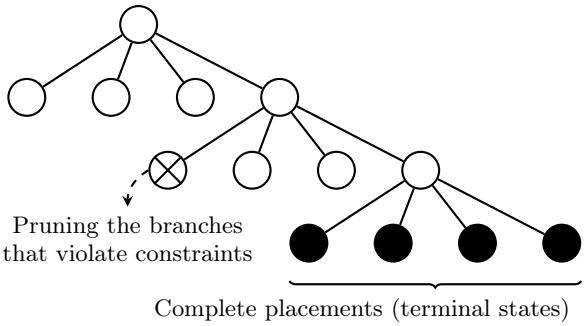

Figure 1. A sample BB search tree, placing a SG with 3 VNFs over a SN with 4 nodes

resources of every single nodes and links). It inherits a list of unplaced VNFs and VLs from its parent state, and it carries a partial placement for a subset of VNFs and VLs of a SG. A terminal state is reached when the placement is finished (*i.e.*, all of the VNFs and VLs of a SG are placed successfully without violating any constraints). A search strategy defines a specific traversal (the order of visiting the states of the tree). This traversal is performed with the help of an ordered list of states, which is called *fringe*. Fringe can be a basic queue (*e.g.*, in Breadth-First Search (BFS)) or a stack (*e.g.*, in Depth-First Search (DFS)) or a priority queue (*e.g.*, in A*).

At the beginning, we create a sorted list of VNFs from the requested SG. The order of the list of VNFs of a SG is determined by traversing the SG in BFS order, starting from the entry point (BFS traversal ensures that our partial placements are connected-subgraphs of the SG, which can be verified against constraint violations). Next, the search starts from the root of the tree by placing the head of the list (the entry point of the SG). We start by instantiating the root state and putting it into the fringe. Then, we enter a search loop. At each iteration, we pop a state from the fringe and check whether it is a terminal state or not. If it is a terminal state, the search terminates. Otherwise, the state is expanded by taking another VNF from the list of VNFs and placing it over each of the SN node. As we place the VNFs, we place a subset of unplaced VLs that now get the chance to be placed into the SN, due to the latest VNF placement (a VL can be placed into SN if its source and destination VNFs are already placed). The expansion results in a list of child states, which are candidate placements of a specific VNF over different SN nodes. The children which does not violate any constraints are added into the fringe to be expanded later. The others are pruned and do not appear later. The search continues until it finds a terminal state and succeeds or reaches a timeout and fails. The depth of the tree is equal to the number of VNFs of the SG, and the branching factor is equal to the number of nodes of the SN.

2) Objectives: To distinguish between different feasible placements of a specific SG over a SN, we need a criteria or an objective. Several objectives are studied in the literature. They can be divided mainly into two categories. The first category belongs to resource-based objectives. They minimize the usage of a particular resource type or a combination of them (regarded as multi-objective). The other category includes objectives that evaluate the overall behavior of the placement strategy and is not necessarily related to specific resource types. Cost of Revenue, Energy Consumption and Service Acceptance (SA) are some of the well-studied objectives in this category [8]. This second category is commonly preferred in the literature, and we are interested in SA, which is defined as the number of services that can be placed successfully over the SN. The objective of SA can be considered as a practical generalization of a resource-based multi-objective strategies.

3) Constraints: We consider CPU and storage as the node resources, bandwidth as the link resource, as well as link latency and end-to-end delay as QoS metrics for the links and the services. Our solution does not impose any limitation for adding more resource types or more QoS metrics for nodes, links or services.

4) A:* A* is among the most popular Artificial Intelligence (AI) Informed Search algorithms that allows traversing the search tree according to f-costs, which are calculated by $f(n) = g(n) + h(n)$ for an arbitrary state of n [12], where 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 a real cost of path from the root to the current state. A* search is proved to be optimal and complete if we choose a heuristic function that is admissible and consistent [12].

B. Search Strategies

We study the influence of five strategies: A* Bandwidth Optimized (ABO), DFS Bandwidth Optimized (DBO), A* and DFS combined Bandwidth Optimized (ADBO), Enhanced Decreasing First-Fit (DFF) (EDFF) and Enhanced Increasing First-Fit (IFF) (EIFF). While EDFF and EIFF concentrate on node-resources, ABO, DBO and ADBO focus on link-resources (*i.e.*, bandwidth). Our ultimate goal is to see how different resource types affect the number of SG that can be accepted. All of the placement candidates of a SG over a SN will eventually use the same amount of SN node-resources, which is equal to the sum of the required resources by its VNFs. Yet, in terms of link-resources, various placement candidates use quite different amount of resources.

1) A Bandwidth Optimized (ABO):* In ABO, we use A* in order to perform a complete and optimal search as fast as possible. Here, the cost is defined as the amount of bandwidth used by the placement. In this case, the fringe is a priority queue. It sorts the states based on f-costs and the head of the queue belongs to the state with minimum f-cost. As a recall, we want to place a SG over the SN optimizing the bandwidth usage. An optimistic heuristic to estimate the cost of placement would be the cost of

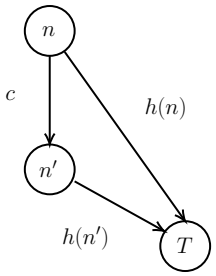

Figure 2. Illustration of triangle inequality on equation 7

placing each unplaced VL over a single SN link instead of a path of several links. Therefore, we consider the total amount of required bandwidth by all of the unplaced VLs as h, and g is the total amount of used bandwidth for all of the placed VLs. Our heuristic is optimistic since it never overestimates the cost to the target state. We need at least one SN link to accommodate each VL. If we cannot place source and destination of a VL over two adjacent SN node, i.e., if we need a path of links to accommodate a VL, the cost would be more what our heuristic had estimated, thus, our heuristic is optimistic.

We can also demonstrate that our heuristic is consistent, or the f-cost would never decrease along a path. In other words, 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' (i.e. $c = g(n') - g(n)$), we need to prove the triangle inequality in 7 illustrated in figure 2.

$$h(n) \leq c + h(n') \tag{7}$$

We know that the estimated cost of placement on n' is less than the estimated cost on n (since, we have less unplaced VLs on n'). We label the estimated cost of transition from n to n' as c'. So, we have $h(n') = h(n) - c'$. Therefore, we just need to prove that $c' \leq c$. Since, we know that our heuristic do not overestimate the cost, this inequality is always true, thus, our heuristic is also consistent. Now, we can assure that our A* search is optimal as well as complete. The optimality of ABO guarantees that if it succeeds to find a placement, no other placement exists using less bandwidth than the found one.

2) DFS Bandwidth Optimized (DBO): In DBO, like ABO, we concentrate on link bandwidth and the cost is defined as the amount of bandwidth used by the placement. We traverse the tree in DFS, so, our fringe will be a stack. After expanding a state, the list of generated states is sorted according to their cost and are put into the fringe. Accordingly, since, the fringe is a stack; the next time that we will pop the fringe, we will expand the child having the lowest bandwidth usage. DBO do not necessarily guarantee optimality or completeness, but it finds a bandwidth-optimized placement faster than ABO. It makes a satisfactory compromise between optimality and speed.

3) A and DFS combined Bandwidth Optimized (ADBO):* ADBO mixes two strategies of ABO and DBO. The search starts finding a placement using ABO. If it finds a placement, the search terminates. Otherwise, if the timeout is reached and ABO could not find a placement, it means that satisfying optimality by ABO would cost us an extra time which is limited in an online-placement. At this point, we give DBO a chance to find a near optimal placement. Since, ADBO uses two strategies sequentially, in order to make a fair evaluation of ADBO along with other strategies, we need to share the given timeout between two strategies (we simply halve the timeout and share it between the two strategies).

4) Enhanced DFF (EDFF) and Enhanced IFF (EIFF): DFF and IFF are two well-known heuristics for service placement. DFF (*aka* Best-Fit) sorts the nodes and places the VNF over the node with the maximum amount of available resources that is adequate to accommodate it. IFF is like DFF but it places the VNF over the node with the minimum amount of available resources. The placement fails to place a service as soon as it does not find any candidate to accommodate a VNF of that SG. We thought how to give these strategies another chance to revise the previously placed VNFs to see how far we can benefit from these heuristics. One of the advantages of BB structure of the search is that it makes it possible to put simply into practice our intuition. By a small tweak in DBO, we can obtain an enhanced version of DFF and IFF heuristics. By sorting the generated children of a state by their amount of available resources instead of their bandwidth usage, we can have EDFF and EIFF (according to whether the sort is ascending or descending). This enhancement allows us to profit from these strategies as much as possible. The evaluations show that EDFF and EIFF can place on average nearly 3-times more SGs than the original ones.

IV. Experimentations

Typically, there are two general approaches to evaluations in the literature. The first one is based on random initialization of the inputs of the evaluation. Reliability and reproducibility is assured by repeating the evaluations with different seeds of randomness. Monte Carlo is a powerful technique, used in this category. Another approach guarantees reliability and reproducibility systematically, also it allows to track the evolution of the results. This is achieved by considering the entire acceptable range of the inputs. In general, although the second approach is more accurate, it is not always feasible, regarding the number of inputs and their wide acceptable range. A practical approximation which facilitates to follow the second approach is dividing the acceptable range into a handful of values. For the reasons of reliability, we follow the second approach. Throughout this section, we choose the parameters for presenting the results by considering the page limitations and results' similarities for different

types and parameters (it is only a brief presentation of a complete investigation on the sheer size of the results).

A. Assumptions

In order to evaluate our search strategies, we necessarily need to consider some assumptions in order to make our evaluations reliable, correct and reproducible (*i.e.,* it is not required to consider these assumptions for the real scenarios; we only need these assumptions in order to make a concentrated and realistic evaluation of our methods). Our assumptions include:

1) No constraints for QoS metrics: We do not consider constraints on QoS metrics in our evaluations. The violations of QoS metrics are systematically detected and taken care in BB search structure. Here, we concentrate on the resources rather than QoS metrics.

2) No constraints for node resources: We performed an interesting evaluation, to reveal the effect of limiting node resources in order to distinguish several placement strategies. We noticed that, by considering low amount of resources for the nodes of the SN, all of our placement strategies act exactly likewise, and each strategy places the same number of services. By increasing this limitation, the effectiveness of different strategies becomes more and more clear. Therefore, limiting node resources at any point, only leads to poor evaluation of the strategies. The results of this evaluation are shown in the Fig. 3.

3) Placing each VNF of a SG in separate SN nodes: We respect the constraint of not placing any two VNFs of a SG over a single SN node on our evaluations. We performed an experiment without this constraint and noticed that the placement problem becomes so simple and relaxed that evaluating different strategies becomes senseless. As it is shown in the Fig. 4, the strategies try to save the resources by placing as many VNFs as possible over a single node in order to minimize bandwidth usage and it continues as long as there is enough node resources left. Additionally, in the production, this constraint is essential in order to satisfy the availability of the services (at least for the services requiring certain degree of availability). By considering redundancy in the service-level and placing the VNFs on separate nodes, we ensure that a node failure would not violate the availability of the services.

B. Parameters and Structure of Evaluations

Our evaluations are executed on a system with Intel Core i7-3687 CPU and 8GB of RAM. The implementations are made in Java, running on Windows 10 OS. All of the evaluations are executed in a single thread. Apparently, their performance could be improved dramatically, regarding parallel execution capability of our strategies. We respect a timeout of 2000 ms in all of our evaluations (*i.e.*, if a strategy could not find a placement within the timeout, it fails). We perform our evaluations over 3 types of SN topologies, which are selected from the dataset of Zoo Topology, named by BT-Europe, BT-Asia-Pacific and BT-North-America with 24, 20 and 36 number of nodes and 74, 62, 152 number of bi-directional links, respectively. We tried three types of topology for SGs (Daisy chain, Ring and Star) with the size of 3 to 8 VNFs and bi-directional VLs. Each VNF requires a single unit of CPU and a single unit of storage and each VL requires a range of bandwidth units from 1 to 10. Each SN node has unlimited resources of CPU and storage and each SN link has 10 units of bandwidth at the beginning of the evaluation. To place a VL over a path in SN, we use Dijkstra's shortest path over hop-count as our routing algorithm for evaluations. Apparently, using sophisticated routing algorithms like SAMCRA [15], which can take QoS metrics into consideration, are preferable for production, but since we concentrate on resources rather that QoS metrics, and we use the routing only for comparing different placement strategies, we tried to choose a basic deterministic routing algorithm. Our evaluations are performed by initializing the entire SN nodes and links with the maximum available resources, then entering a loop. On each iteration, a SG is created based on a topology, size, amount of required resources of VNFs and VLs. Then, we try to place this SG over the SN using a strategy within a time limit. If the strategy succeeded to find a placement, then the placement is applied to the SN by updating the remaining resources then we start a new iteration. Otherwise, we terminate the evaluation and report the number of SGs that we have been able to place.

C. DFF and IFF vs. Their Enhanced Versions

Figure 5 compares DFF and IFF heuristics vs. their enhanced versions, placing different sizes of SGs of daisy chain topology (each VL requiring 1 unit of bandwidth) over BT-Europe network. As shown, this enhancement allows us to place on average nearly 3-times more SGs than the original DFF and IFF.

D. The Idea Behind ADBO

Table II demonstrates the number of placed SGs as well as the percent of the total remained bandwidth on SN at the end of the placement for ABO, DBO and ADBO strategies. We place different sizes of SGs of daisy chain topology (each VNF requiring 1 unit of cpu and storage and each VL, 1 unit of bandwidth) over BT-Europe network. It is highly satisfactory that by placing SGs with 6 VNFs, ABO can place more SGs than DBO (60 vs. 57), besides, leaving more remaining bandwidth for SN (18.92% vs 12.97%). But, regarding the placement of SGs with 8 VNFs, although ABO can leave a valuable amount of SN bandwidth untouched (43.24% vs 7.03%), it places lower number of SGs (30 vs 38), i.e., time is sacrificed over optimality. Considering the entire results of evaluations, we came to this idea that we can benefit from combining ABO and DBO, introducing ADBO which shares the timeout between ABO and DBO and executes DBO after ABO consecutively.

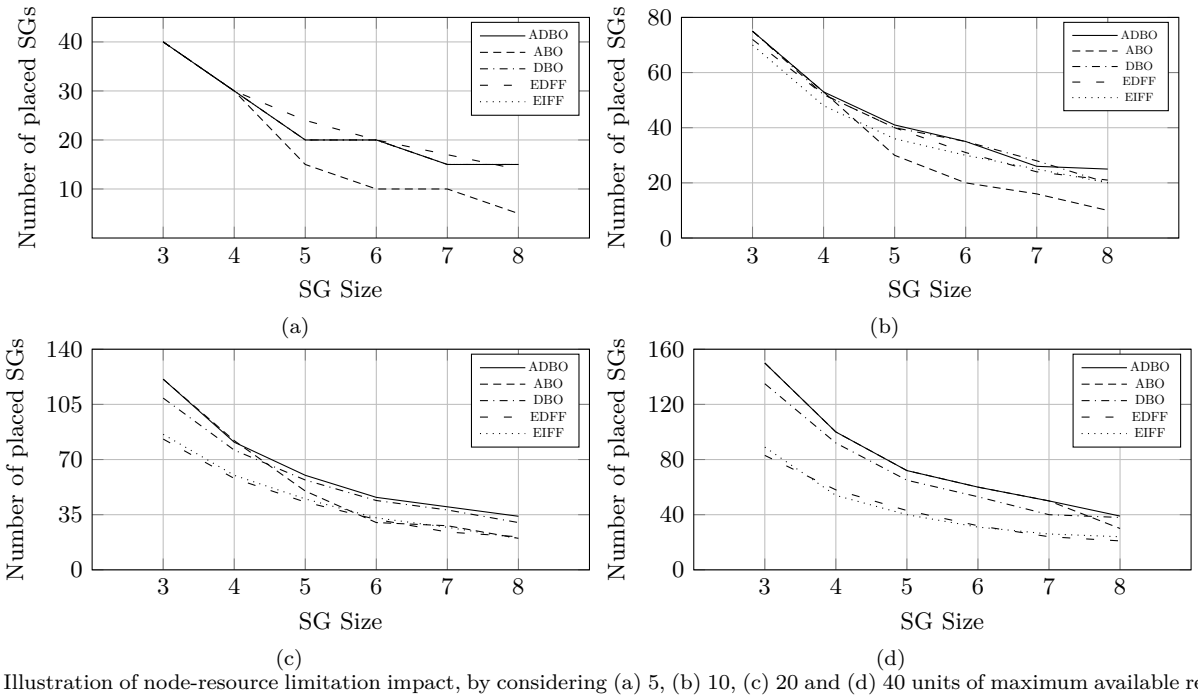

Figure 3. Illustration of node-resource limitation impact, by considering (a) 5, (b) 10, (c) 20 and (d) 40 units of maximum available resources

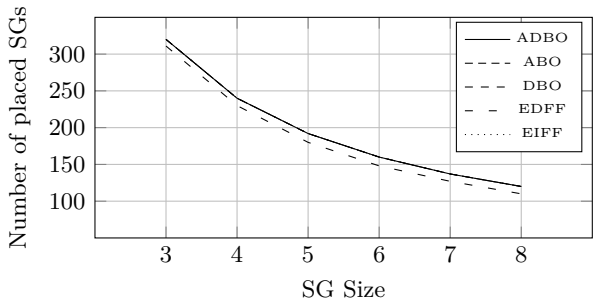

Figure 4. Illustration of being able to place multiple VNFs over a single SN node impact

Table II
The idea behind ADBO

SG Size	Number of placed SGs			% of remained bandwidth		
	ABO	DBO	ADBO	ABO	DBO	ADBO
3	180	173	180	2.70	2.16	2.70
4	115	101	115	5.41	9.46	5.41
5	78	78	78	13.51	8.38	13.51
6	**60**	**57**	60	**18.92**	**12.97**	18.92
7	55	46	55	6.76	12.16	6.76
8	**30**	**38**	39	**43.24**	**7.03**	17.57

E. Service Acceptance

Comprehensive evaluations has been carried out. We performed 900 evaluations for each SN type, considering different types and parameters. By having 3 types of SN, it makes 2700 evaluations altogether. It is observed that ADBO is almost always superior to other strategies, making us able to measure other strategies compared to ADBO (Table III). We utilize quartiles (Q1, Q2 (Median), Q3) as well as min-avg-max, in order to better represent the results. Note that, an improvement of 400% means that ADBO could place 5 times more SGs having the same evaluation parameters in comparison to the related strategy. Beyond this satisfactory results in terms of average, it is quite rewarding to witness that ADBO could place 24 to 35 times more SGs (regarding max in the figure), when the placement problem becomes so complicated that DBO, EDFF and EIFF could not find any placement or they could only find 1 or 2.

Table III
Service Acceptance Improvements, gained by using ADBO instead of other strategies (%)

Strategy	Q1	Q2	Q3	Min	Average	Max
ABO	0	0	0	0	**7**	110
DBO	0	8	33	-33	**128**	3400
EDFF	60	100	125	-28	**184**	2300
EIFF	67	92	125	0	**211**	3400
DFF	200	300	500	60	**429**	2300
IFF	300	500	700	100	**564**	2200

F. Execution Time and Complexity

Each strategy places SGs as long as it succeeds in accepting them. The placements fail and are terminated when no solution is possible or when the time limit elapses. Except the last placement, which fails within the timeout, others are executed in a relatively short time. Although, it depends on the size of the SG and SN, on average, ADBO, ABO, DBO, EDFF and EIFF could place a service in 53, 52, 40, 43 and 46 ms respectively, according to our evaluations.

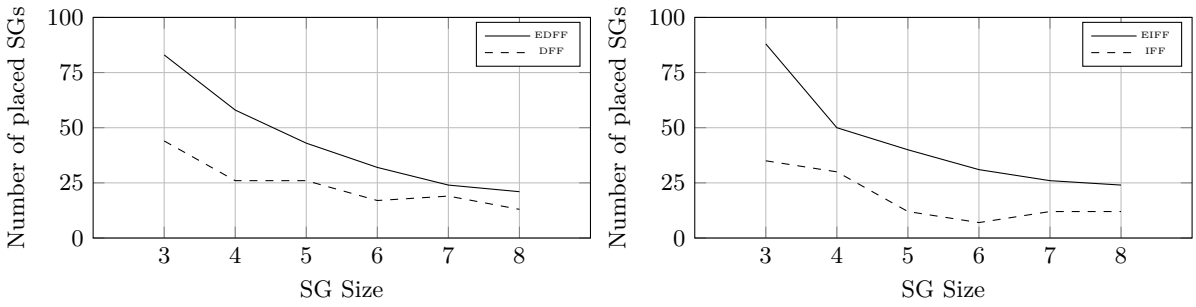

Figure 5. EDFF and EIFF vs. DFF and IFF

In terms of the complexity, considering that the placement problem is NP-Hard, clearly, the complexity of our strategies is not polynomial. The complexity of a BB method heavily depends on the number of expanded states, which is determined by the traversal strategy. While, the complexity of these strategies grows exponentially, they do not grow equally (e.g., take the complexity of $O(e^n)$ vs. $O(e^{0.01n})$. If we remove the heuristic component of A*, we get Uniform-Cost Search (UCS). Using UCS instead of A* in ABO, gives us an intuition about the complexity. In general, an informed search strategy, if the heuristic respects the triangle inequality, in the worst case, acts better than an optimal uninformed search strategy like UCS. By evaluations, we noticed that obtaining optimal results by UCS is not even possible for more than 90% of our cases, even by giving it more time. The results witness the incomparable superiority of ABO and the effectiveness of our heuristic.

V. Conclusion

Although the VNF placement problem has been around for several years, but the need for a solution that could compromise optimality and scalability to a certain degree still exists. Formerly, the researches were mainly concentrated on ILP formulations or heuristic greedy search. They were trying whether to guarantee optimal results or to meet scalability and time constraint requirements. Recently, sophisticated AI and ML techniques have gained attention, regarding their capability to bring optimality and scalability close to each other. We proposed a solution based on a BB structure, leveraging AI search strategies, especially A*, using an admissible and consistent heuristic, which is capable of finding optimal placements, relatively fast. Sizeable empirical analysis has been performed and the results confirm a considerable improvement (429% on average) in comparison with the popular heuristic placement method of Best-Fit.

Investigating the effects of QoS metrics over the VNF placement problem, and exploring SNs with Edge/Core topologies that are short on edge network resources, are some open issues for future direction.

References

[1] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte. Orchestrating virtualized network functions. *IEEE TNSM*, 13(4):725–739, 2016.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[3] X. Fu, F. R. Yu, J. Wang, Q. Qi, and J. Liao. Dynamic service function chain embedding for nfv-enabled iot: A deep reinforcement learning approach. *IEEE Transactions on Wireless Communications*, 19(1):507–519, 2019.

[4] M. A. Khoshkholghi, J. Taheri, D. Bhamare, and A. Kassler. Optimized service chain placement using genetic algorithm. In *2019 IEEE Conference on Network Softwarization (NetSoft)*, pages 472–479. IEEE, 2019.

[5] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 98–106. IEEE, 2015.

[6] D. M. Manias, H. Hawilo, M. Jammal, and A. Shami. Depth-optimized delay-aware tree (do-dat) for virtual network function placement. *IEEE Networking Letters*, 2(3):149–153, 2020.

[7] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[8] G. Mirjalily and L. Zhiquan. Optimal network function virtualization and service function chaining: A survey. *Chinese Journal of Electronics*, 27(4):704–717, 2018.

[9] 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*, 38(2):263–278, 2019.

[10] T. Pham, J.-M. Sanner, C. Morin, and Y. Hadjadj-Aoul. Virtual network functionforwarding graph embedding: A genetic algorithm approach. *Int. J. of Communication Systems*, 33(10):e4098, 2020. e4098 0.1002/dac.4098.

[11] P. Quang, Y. Hadjadj-Aoul, and A. Outtagarts. A deep reinforcement learning approach for vnf forwarding graph embedding. *IEEE TNSM*, 16(4):1318–1331, 2019.

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

[13] J. M. Spivey and J. Abrial. *The Z notation*. Prentice Hall Hemel Hempstead, 1992.

[14] 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*, 10(11):646, 2018.

[15] P. Van Mieghem and F. A. Kuipers. Concepts of exact qos routing algorithms. *IEEE/ACM TON*, 12(5):851–864, 2004.

[16] O. A. Wahab, N. Kara, C. Edstrom, and Y. Lemieux. Maple: A machine learning approach for efficient placement and adjustment of virtual network functions. *JNCA*, 142:37–50, 2019.