Shortening the Deployment Time of SFCs by Adaptively Querying Resource Providers – Extended Version

Ali El Amine, Olivier Brun, Slim Abdellatif, Pascal Berthou

Shortening the Deployment Time of SFCs by Adaptively Querying Resource Providers

Ali El Amine*, Olivier Brun†, Slim Abdellatif‡ and Pascal Berthou§
LAAS-CNRS, INSA, Université Paul Sabatier, Université de Toulouse, CNRS, Toulouse, France
Email: *aelamine@laas.fr, †brun@laas.fr, ‡slim@laas.fr, §berthou@laas.fr

***Abstract*—We consider the SFC embedding (SFCE) problem in the Slice as a Service (SlaaS) model. In this model, a slice provider leases resources from multiple cloud and network providers in order to instantiate the Service Function Chain (SFC) requested by a slice tenant. As the slice provider has no visibility on the infrastructures of the resource providers, in which resources may be purchased and released quite rapidly, it has to query them to determine what are the possible allocations and their costs. We show that when there are many resource providers and many VNFs composing the SFC, the number of queries to be made for discovering a minimum cost SFC embedding grows quickly, leading to excessively long deployment times. In order to reduce the latter quantity, we propose to query resource providers strategically, rather than collecting the information on all possible allocations at once. We provide bounds on the number of queries to be made in this approach, and propose to exploit a *Shortest Path Discovery* algorithm in order to reduce this number of queries and thus the SFC deployment time. Our numerical results suggest that this algorithm is fairly efficient, in particular when initial estimates of allocation costs can be provided by the slice provider, and that the deployment times can be significantly shortened.**

***Index Terms*—Service Function Chain, Virtual Network Function, Slice as a Service, Shortest Path Discovery**

I. Introduction

A SFC is a series of network functions (e.g., traffic optimizers, firewalls or Web proxies), which a packet must flow through. With the emergence of software-centric networking technologies such as NFV and SDN, these network functions, which were traditionally implemented on hardwired middleboxes, can now be run in datacenters and operated as cloud services. This allows SFCs to be deployed flexibly on request and according to demand by dynamically composing Virtual Network Functions (VNFs) [1], [2]. In Fig. 1, we illustrate an example of a SFC characterized by an ingress node, an egress node and a sequence of three VNFs. The edges connecting VNF nodes represent virtual links.

The SFCE problem amounts to mapping the logical service network corresponding to a SFC to existing network capabilities. It includes the placement of VNFs on a shared computing substrate usually composed of multiple clouds, as well as the establishment of the virtual links between the VNFs in the substrate networks. The solution should meet the resource demands of the SFC at minimum cost, and possibly some other technical constraints. For instance, some VNFs may have to be deployed in private clouds for privacy or security reasons, or are location dependent (e.g., proxies and caches should be placed in proximity to end users).

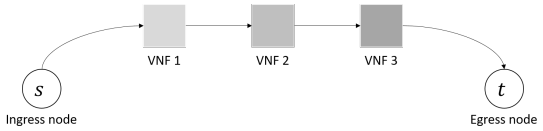

Fig. 1: A service chain example.

Existing work on the SFCE problem assume that the substrate infrastructures are known (taken as input) and seek to optimize some infrastructure-related metrics. For instance, Elias et al. [3] formulate the SFCE problem as a non-linear optimization problem in which the goal is to minimize the congestion of physical resources. Similarly, the authors in [4] seek to determine the required number and placement of VNFs that optimize network operational costs and utilization, without violating service level agreements. On the other hand, the authors in [5] designed a mathematical model that allows a scalable exact solution scheme to solve the SFCE problem. Other approaches fall into the category of heuristic-based solutions. For example, the work in [6] proposes the SFC-MAP algorithm, that selects and places VNFs to minimize the embedding costs of SFC requests. Similarly, the work in [7] proposes a heuristic solution that iteratively places the VNFs in series using the nearest search procedure. Some work relied om machine learning and in particular on reinforcement learning to efficiently solve the SFCE problem [8].

In this work, we define a slice as a SFC with specific requirements in terms of computing and storage resources and network Quality of Service (QoS), and study the SFCE problem in the context of the SlaaS model. As shown in Figure 2, there are three main players under this model. The first one is the *slice tenant* who requests a SFC to be deployed. The slice tenant's request is described by a slice Template/Descriptor similar to the Network Slice Template (NST) specified by ETSI in [9]. This request is forwarded to a *slice provider*, which leases resources from *resource providers* in order to establish the requested SFC at minimum cost. Two categories of resource providers are considered: on one hand, cloud providers offering the compute, storage and network resources required to execute VNFs, and on the other hand, network connectivity providers which can establish virtual links between geographically distant clouds. The slice provider queries the *resource providers* in order to discover the different options of available resources that will fulfil the

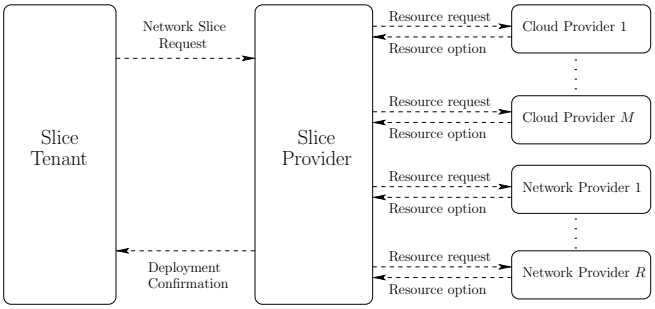

Fig. 2: Slice as a Service model.

tenant's request requirements at minimum cost. When queried, a resource providers replies with *resource options* comprising pricing and resource details. Once all resource options have been fetched by the slice provider, the latter chooses among the different options one of those satisfying the SFC constraints at minimum cost, deploys the SFC, and then informs the slice tenant of the deployment.

In contrast to previous work, we assume that the slice provider has no knowledge about the infrastructures of the resource providers, let alone the level of their resource usage. Also, it has to decide where to place the VNFs and how to interconnect them so as to meet the resource requirements of the SFC at minimum cost. The issue is that, as resources are dynamically purchased or released in the clouds, the slice provider does not know whether a cloud will be able to accommodate a VNF, let alone at what cost. Similarly, it does not know whether a connectivity provider will be able to offer the required connectivity between two VNFs, nor at what price. So the slice provider has to query resource providers to determine what are the possible allocations and their costs. It turns out, that in practice, as discussed in Section II, the time required to collect the information on all resource options is much larger than the time required to compute a minimum cost solution once this information available.

To the extend of our knowledge, the present paper is the first one to investigate how the number of queries, and hence the deployment time of SFCs, could be reduced in the SlaaS model. Its main contribution is to show that the deployment time could be greatly shortened by querying resource providers adaptively, rather than naively collecting the information on all resource options at once. We provide bounds on the number of queries that need to be made to the resource providers for solving the SFCE problem and propose to exploit an algorithm proposed for the *Shortest Path Discovery* problem in order to reduce this number of queries (and thus the deployment time) without questioning the optimal embedding. Our numerical results suggest that this algorithm is fairly efficient, in particular when initial estimates of allocation costs can be provided by the slice provider.

The paper is organized as follows. Section II is devoted to the mathematical formulation of the problem. We review some known results on the Shortest Path Discovery problem in Section III, where we also obtain some bounds on the number of queries in the SFCE problem. Numerical results are presented in Section IV and some conclusions are drawn in Section V.

II. Problem Statement

In this paper, we focus on mapping a SFC on the physical resources furnished by multiple resource providers. Let s and t be the source and destination nodes of the service chain, which is represented as an ordered sequence of K VNFs $(f_1, f_2, \ldots, f_K)$, with some functions possibly repeated. There are M public or private Clouds and for each VNF f_k, we are given a set $\mathcal{D}_k \subseteq \mathcal{M} = \{1, \ldots, M\}$ of Clouds where it can be executed. We let $N > M$ be the total number of resource providers and define $\mathcal{N}_{i,j} \subseteq \mathcal{N} = \{1, \ldots, N\}$ as the set of resource providers offering connectivity between Clouds i and j. We assume that if two consecutive VNFs are executed in the same Cloud i, the connectivity between them is provided by the Cloud provider, so that $\mathcal{N}_{i,i} = \{i\}$. In contrast, the connectivity between VNFs running in different Clouds i and $j \neq i$ is furnished by some of the $R = N - M$ network providers, so that $\mathcal{N}_{i,j} \cap \mathcal{M} = \varnothing$. Similarly, $\mathcal{N}_{s,i}$ (resp. $\mathcal{N}_{j,t}$) is defined as the set of network providers offering connectivity between the source node s (resp. Cloud $j \in \mathcal{M}$) and Cloud $i \in \mathcal{M}$ (resp. the destination node t).

For the above definitions to be consistent, we shall further assume that if VNF f_k can be deployed in Cloud i and VNF f_{k+1} can be deployed in Cloud $j \neq i$, then the connectivity between those Clouds can be provided by at least one network provider, that is, $\mathcal{N}_{i,j} \neq \varnothing$. Similarly, we assume that $i \in \mathcal{D}_1$ (resp. $j \in \mathcal{D}_K$) implies that $\mathcal{N}_{s,i} \neq \varnothing$ (resp. $\mathcal{N}_{j,t} \neq \varnothing$).

Figure 3 depicts the setting considered with a simple example in which packets sent by source node s to destination node t have to go through VNFs f_1, f_2 and f_3, in this order. In this example, we have $\mathcal{D}_1 = \{1, 2\}$, $\mathcal{D}_2 = \{1, 2, 3\}$ and $\mathcal{D}_3 = \{2, 3\}$. Connectivity between the clouds can be provided by two different network providers. There are some asymmetries in the connectivity pattern however, since Cloud 1 can only be reached from Network A, whereas Clouds 2 and 3 are connected to both networks A and B. Similarly, the source node s is connected to both networks, whereas the destination node t can only be reached via network B.

We shall assume that the resource requirements of each VNF f_k in the chain, as well as the resource requirements for the virtual links interconnecting the VNFs, are known beforehand. However, we assume that the slice provider does not know the infrastructures of the resource providers and that it has to query them to determine what are the possible allocations and their costs. We note that the resource options proposed by the resource providers may have different technical characteristics, in addition to their price. In all cases, we assume that the slice provider is able to compare different resource options and to assign a unique cost to each one. We assume that this cost is infinite when a resource providers cannot provide the requested resources.

Once the possible allocations and their costs are known,as observed in [10], the SFCE problem can then be cast as

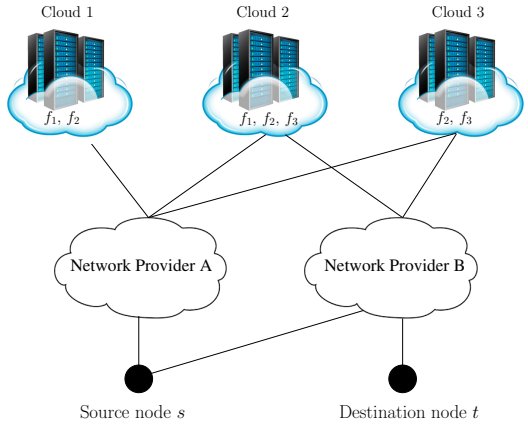

Fig. 3: Example scenario with $K = 3$ VNFs, $M = 3$ Clouds and $R = 2$ interconnection networks.

that of finding a shortest $s - t$ path in a layered directed multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. As illustrated in Figure 4, the set of nodes is $\mathcal{V} = \bigcup_{k=0}^{K+1} \mathcal{V}_k$, where $\mathcal{V}_0 = \{s\}$, $\mathcal{V}_{K+1} = \{t\}$ and $\mathcal{V}_k = \{v_{k,j} \ : \ j \in \mathcal{D}_k\} \cup \left\{v'_{k,j} \ : \ j \in \mathcal{D}_k\right\}$ for $k = 1, \ldots, K$. The nodes $v_{k,j}$ and $v'_{k,j}$ are used to represent the allocation of VNF f_k to Cloud $j \in \mathcal{D}_k$. The set of edges $\mathcal{E}$ is composed of a directed edge $e = (v_{k,j}, v'_{k,j})$ for all $k \in \{1, \ldots, K\}$ and all $j \in \mathcal{D}_k$, whose cost c_e represents the cost of running VNF f_k in Cloud $j \in \mathcal{D}_k$ ($c_e = +\infty$ if Cloud j cannot accommodate VNF f_k). The other edges represent the interconnection possibilities between VNFs and the associated costs, which again shall be assumed to be $+\infty$ when a virtual link cannot be afforded. There is a directed edge from node $v'_{k,j}$ to node $v_{k+1,j}$ whenever $j \in \mathcal{D}_k \cap \mathcal{D}_{k+1}$. This edge represents the connectivity between VNFs f_k and f_{k+1} within Cloud j and its cost is the cost of setting up a virtual link with the desired characteristics in this Cloud. For each network provider n, there is also an edge from node $v'_{k,i}$ to node $v_{k+1,j}$ associated to this network provider whenever $n \in \mathcal{N}_{i,j}$, for all $i, j \in \mathcal{M}$, $j \neq i$. Finally, for each network provider n and each Cloud $j \in \mathcal{D}_1$, there is a directed edge from s to $v_{1,j}$ if $n \in \mathcal{N}_{s,j}$, and similarly there is a directed edge from $v'_{K,j}$ to t for each Cloud $j \in \mathcal{D}_K$ such that $n \in \mathcal{N}_{j,t}$.

Note that the above model assumes linear costs. Hence, if two or more VNFs are allocated to the same cloud, the costs just add up, i.e., there is no economies of scale for running several VNFs in the same cloud. Similarly, if two or more virtual links are established between the same pair of clouds, they cannot be "aggregated", and the costs add up as well in this case. The latter assumption is particularly justified when the virtual links have different QoS requirements.

If the costs of the edges in the graph $\mathcal{G}$ were known, solving the SFCE problem described above would be as simple as finding a shortest $s-t$ path in $\mathcal{G}$. The issue is that, as explained above, these costs are not known by the slice provider and have to be discovered by querying the resource providers. The total number of queries to be made corresponds to the number of edges in the graph $\mathcal{G}$, which is

TABLE I: Number of queries as a function of network configuration.

Parameters	# of queries		
	$K = 5$	$K = 10$	$K = 15$
$M = 3$, $R = 3$	117	237	357
$M = 5$, $R = 3$	315	665	1015
$M = 7$, $R = 5$	973	2093	3213

$$|\mathcal{E}| = \beta_0 + \sum_{k=1}^{K} (n_k + \beta_k) \,, \tag{1}$$

where $n_k = |\mathcal{D}_k|$ is the number of possible placements for VNF f_k. The quantity $\beta_k = \sum_{i \in \mathcal{D}_k} \sum_{j \in \mathcal{D}_{k+1}} |\mathcal{N}_{i,j}|$ represents the total number of virtual links that can be established for connecting the potential locations for VNF f_k with the potential locations for VNF f_{k+1}, for $k = 1, \ldots, K-1$. Similarly, $\beta_0 = \sum_{i \in \mathcal{D}_1} |\mathcal{N}_{s,i}|$ represents the total number of edges outgoing from node s, whereas $\beta_K = \sum_{i \in \mathcal{D}_K} |\mathcal{N}_{i,t}|$ represents the total number of edges incoming to node t. For the simple example of Figure 3, it already yields $|\mathcal{A}| = 27$ queries to be made to the resource providers. More generally, in the symmetric case where all VNFs can potentially be deployed in all Clouds and where all the $R = N - M$ network providers can establish virtual links between any pair of Clouds, we have $\beta_0 = \beta_K = R\,M$ and $\beta_k = M\,(1 + R\,(M-1))$ for $k = 1, \ldots, K-1$, and the total number of queries to be made is $|\mathcal{E}| = M\,\{2(R+K) - 1 + (K-1)R(M-1)\}$. Table I shows the required number of queries in this symmetric case for some network configurations. As should be apparent from the values in Table I, in practice the cost of solving the SFCE problem will be significantly dominated by the cost of collecting the edge costs of the layered graph $\mathcal{G}$, which of course has a direct impact on the deployment time of the SFC (e.g., the time required to collect all edge costs is almost 2 minutes in the simple case $K = 5$, $M = 3$ and $R = 3$, assuming each request requires 1 second). It follows that the main lever for reducing the deployment time of service chains in the SlaaS model is to reduce the number of queries made to the resource providers. As discussed in Section III, this problem perfectly falls in the scope of the so-called *Shortest Path Discovery* (SPD) problem introduced in [11].

III. Shortest Path Discovery Problem

In the SPD problem, we are given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a function $c : \mathcal{E} \to \mathbb{R}_+$ assigning to each edge $e \in \mathcal{E}$ a cost $c_e > 0$, and two distinguished vertices $s, t \in \mathcal{V}$ such that at least one directed path from s to t exists in $\mathcal{G}$. As in the classical shortest path problem, one seeks for a path connecting s and t with the least cost, where the cost of a path π is defined as $c(\pi) = \sum_{e \in \pi} c_e$. The issue is that the edge costs are initially unknown. However, their values can be discovered by querying an *Oracle*. The goal is therefore to discover a shortest $s - t$ path with the minimum number of queries to the Oracle. In order to do so, we may have some initial knowledge in the form of a function $c^{(0)} : \mathcal{E} \to \mathbb{R}_+$

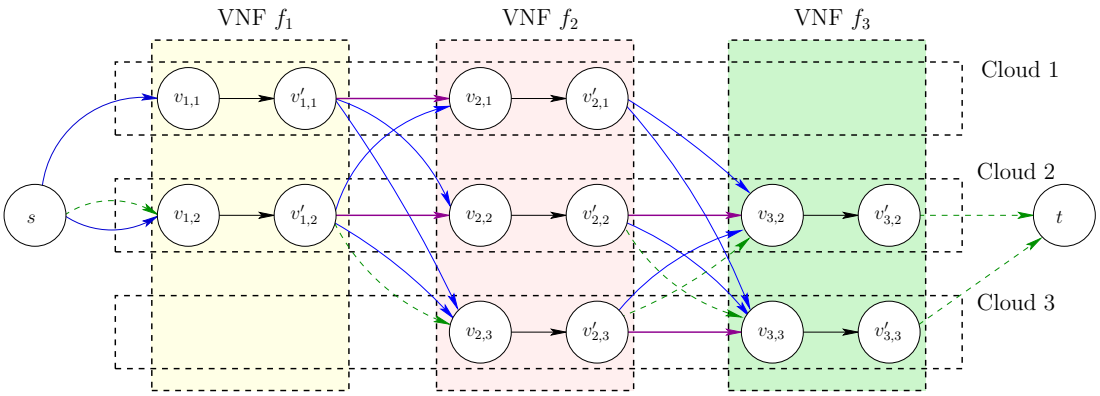

Fig. 4: Layered multigraph for the example of Figure 3. Blue solid edges represent virtual links provided by network provider A, whereas green dashed edges are those provided by network provider B. Thick solid edges in violet represent virtual links established within a datacenter by a Cloud provider.

providing estimates of the edge costs such that $c_e^{(0)} \leq c_e$ for all edges $e \in \mathcal{E}$. When $c_e^{(0)} = 0$ for all edges $e \in \mathcal{E}$, there is no initial knowledge and we say that the edge costs are *totally unknown*.

We review some known results for the SPD problem in Section III-A, and provide bounds on the number of queries for the SFCE problem in Section III-B.

A. Known results on the SPD problem

Given an instance $P = (\mathcal{G}, c)$ of the SPD problem, and possibly some initial cost estimates $c^{(0)}$, any algorithm $\mathcal{A}$ for solving SPDs must propose when it terminates a path π from s to t, and be able to certify that this path is of minimal cost. This clearly implies that the algorithm has to discover the cost of some of the edges by querying the Oracle. For short, we shall say that the algorithm uncovers an edge when it queries the Oracle for its cost. Let $\mathcal{C}_\mathcal{A} \subseteq \mathcal{E}$ be the set of edges uncovered by algorithm $\mathcal{A}$ when it terminates[1]. Following [12], we say that $\mathcal{C}_\mathcal{A}$ is a certificate for path π if $\pi \subseteq \mathcal{C}_\mathcal{A}$ and if π is a shortest $s-t$ path in the graph G with edge costs

$$c'_e = \begin{cases} c_e & \text{if } e \in \mathcal{C}_\mathcal{A}, \\ c_e^{(0)} & \text{otherwise.} \end{cases} \tag{2}$$

As proved in [11], a fundamental property of any correct algorithm $\mathcal{A}$ for solving SPDs is that the set of edges it uncovers is a certificate for the $s-t$ path it proposes when it terminates. An algorithm $\mathcal{A}$ is optimally effective on instance $P = (\mathcal{G}, c)$ if for any other algorithm $\mathcal{A}'$ it holds that $|\mathcal{C}_\mathcal{A}| \leq |\mathcal{C}_{\mathcal{A}'}|$, that is, if the number of edges uncovered by $\mathcal{A}'$ is not smaller than the number of edges uncovered by $\mathcal{A}$. The algorithm $\mathcal{A}$ is optimally effective if the above inequality holds on all problem instances. As observed in [11], a direct consequence of the previous property is that an "optimally effective algorithm can and should stop when the shortest path w.r.t. the best estimate of the cost has no unknown edges".

[1]The set $\mathcal{C}_\mathcal{A}$ of uncovered edges obviously depends on the problem instance P and on the initial estimates $c^{(0)}$. We do not make this dependance explicit in order to simplify notations.

Building on this observation, the authors of [11] propose a simple algorithm for solving SPDs, see Algorithm 1 below. The algorithm maintains the set $\mathcal{C}$ of uncovered edges as well as the best estimate of the edges costs c'_e, which are updated using (2). At each iteration, the algorithm computes a shortest path π from s to t using the cost estimates c'_e, and then uncovers the unknown edges of this path if any, before updating the estimates c'_e and the set of uncovered edges $\mathcal{C}$. The algorithm stops when the cost of all edges of the shortest path π are known. This algorithm is termed greedy because it uncovers *all* edges of the best path w.r.t the weights c'_e. As observed in [11], it should be possible to reduce further the number of queries by uncovering those edges one by one, in some strategic order.

Algorithm 1 The Greedy SPD Algorithm

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $s, t \in \mathcal{G}$, c and $c^{(0)}$
  $\mathcal{C} = \varnothing$
  $c' = c^{(0)}$
  repeat
    $\pi \leftarrow$ Shortest path from s to t with weights c'
    $n = |\pi \setminus \mathcal{C}|$
    if $n > 0$ **then**
      Query all edges $e \in \pi \setminus \mathcal{C}$
      Set $c'_e = c_e$ for all edges $e \in \pi \setminus \mathcal{C}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \pi$
    end if
  until $n = 0$
  return π

Using the example in Figure 4 and assuming some values of the edge costs c_e and of the initial estimates $c_e^{(0)}$, Figure 5 illustrates how the Greedy SPD algorithm works. By exploiting the initial knowledge on the edge costs, the algorithm is able to uncover only 12 out of the 27 edges.

Up to now, it has been assumed that the initial estimates provide lower bounds on the true costs, that is, $c_e^{(0)} \leq c_e$. As shown in [11], this restriction can be removed for integer-

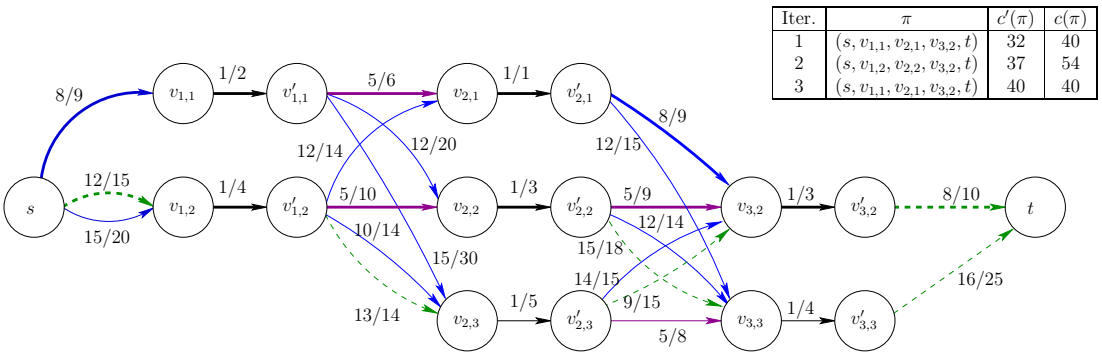

Fig. 5: Illustration of the Greedy SPD algorithm for the layered multigraph of Figure 4. The values $c_e^{(0)}/c_e$ are shown next to the edges. The thick edges are those that are uncovered by the algorithm. The table indicates the best path π found at each iteration (only the main intermediate points are indicated, as the others are obvious), as well as its estimated cost $c'(\pi)$ and its true cost $c(\pi)$.

valued costs, provided that the estimates c'_e are updated as follows

$$c'_e = \begin{cases} c_e^{(0)} + c_e\, L_0 & \text{if edge } e \text{ is uncovered,} \\ c_e^{(0)} & \text{otherwise,} \end{cases} \tag{3}$$

where $L_0 \geq \max_\pi c^{(0)}(\pi)$ is an upper bound on the cost of any $s-t$ path w.r.t. the initial cost estimates $c_e^{(0)}$.

B. Bounds on the number of queries for the SFCE problem

In the following, we shall consider the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ associated to an instance of the NSE problem. This graph has the structure described in Section II. We recall that n_k is the number of directed edges between nodes $u, v \in \mathcal{V}_k$ for $k = 1, \ldots, K$, whereas β_k is the number of edges between nodes $u \in V_k$ and nodes $v \in V_{k+1}$ for $k = 0, \ldots, K$. As we have assumed that $\mathcal{N}_{i,j} \neq \varnothing$ for all $i \in \mathcal{D}_k$ and all $j \in \mathcal{D}_{k+1}$ and for all $k = 1, \ldots, K-1$, and also that $\mathcal{N}_{j,t} \neq \varnothing$ for all $j \in \mathcal{D}_K$, we have $n_k \leq \beta_k$ for all $k = 1, \ldots, K$. Similarly, the assumption that $\mathcal{N}_{s,i} \neq \varnothing$ for all $i \in \mathcal{D}_1$ implies that $\beta_0 \geq n_1$.

Exploiting the specific structure of the graph $\mathcal{G}$, we first obtain a lower bound on the number of queries to be made when the edge costs are totally unknown.

Lemma 1. *In the NSE problem with no previous knowledge, the number of requests to be made to the resource providers is at least* $2K + \min_{k=1,\ldots,K}(n_k)$.

Proof. Consider an instance of the NSE problem and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the associated directed graph. Let $\mathcal{A}$ be an SPD algorithm, and let π be the path it proposes and $\mathcal{C}_\mathcal{A}$ be its certificate. Note that $\pi \subseteq \mathcal{C}_\mathcal{A}$ implies that $c(\pi) > 0$. The cardinal of $\mathcal{C}_\mathcal{A}$ represents the number of requests made using algorithm $\mathcal{A}$.

We first show that $\mathcal{C}_\mathcal{A}$ contains a cut-set in $\mathcal{G}$ such that the corresponding cut places s in one set of the partition and t in the other. The proof is by contradiction. Assume that $\mathcal{C}_\mathcal{A}$ does not contain such a cut-set. This implies that there exists at least one $s-t$ path $\pi' \subseteq \mathcal{E} \setminus \mathcal{C}_\mathcal{A}$. As the edges of π' have not been uncovered, nothing is known about their costs, and we have $c(\pi') = \sum_{e \in \pi'} c_e^{(0)} = 0 < c(\pi)$, which is a contradiction because π is supposed to be of minimum cost. We thus conclude that $\mathcal{C}_\mathcal{A}$ contains a cut-set separating s and t.

Now, observe that the smallest $s-t$ cut-set in graph $\mathcal{G}$ has size $\min_{k=1,\ldots,K}(n_k)$. Indeed, we know that $\beta_k \geq n_k$ for all $k = 1, \ldots K$. We also know that $\beta_0 \geq n_1$. Assuming that each edge has capacity 1, the structure of the graph $\mathcal{G}$ then implies that the maximum $s-t$ flow is $\min_{k=1,\ldots,K}(n_k)$, which implies that the size of a minimum cut-set is as stated.

The result then follows from the fact that the path π, which is of size $2K+1$, is included in $\mathcal{C}_\mathcal{A}$, and by observing that π and any cut-set have at most one edge in common. □

In the symmetric case, Lemma 1 implies that the number of queries to be made is at least $2K + M$. Note however that the lower bound of Lemma 1 is optimistic, as in practice many more queries are needed. In fact, there exists a bad instance for which any SPD algorithm needs to uncover *all* edges, as proven in Lemma 2 below.

Lemma 2. *In the NSE problem with no previous knowledge, there exists a bad instance for which the number of requests made to the resource providers by any SPD algorithm* $\mathcal{A}$ *is* $\beta_0 + \sum_{k=1}^{K}(n_k + \beta_k)$.

Proof. Consider an instance of the NSE problem and let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the associated directed graph. Choose an arbitrary $s-t$ path in $\mathcal{G}$ and assume that the edge costs are fixed as follows. For any edge $e \in \pi$, $c_e = 1$, whereas for all other edges $e \in \mathcal{E} \setminus \pi$, $c_e = 1 + \epsilon$, where $\epsilon < \frac{1}{2K}$. It follows that π is of cost $c(\pi) = 2K + 1$, and that it is the only shortest $s-t$ path in $\mathcal{G}$, as all other paths have a cost greater than or equal to $2K + 1 + \epsilon$ (this lower bound being reached if a single edge of π is replaced by a parallel edge). Given a correct SPD algorithm $\mathcal{A}$, it should return the path π for this instance, and uncover a set of edges $\mathcal{C}_\mathcal{A}$ allowing to certify that π is indeed a shortest $s-t$ path. Assume that $\mathcal{C}_\mathcal{A} \neq \mathcal{E}$. As $\pi \subset \mathcal{C}_\mathcal{A}$, it follows that there exists at least one edge $e \notin \pi$ for which nothing is known. Consider an $s-t$ path π' such

that $e \in \pi'$. The estimated length $c'(\pi')$ of this path is at most $2K(1+\epsilon) < c(\pi)$. This implies that $\mathcal{C}_{\mathcal{A}}$ is not a certificate for path π , which is a contradiction. Hence, $\mathcal{C}_{\mathcal{A}} = \mathcal{E}$ for any SPD algorithm $\mathcal{A}$. The result then follows from (1). □

IV. Numerical Results

In this section, we experimentally evaluate by how much the deployment time of a service chain can be shortened using the Greedy SPD algorithm. We first describe the performance metrics to be evaluated in Section IV-A and the procedure used for the random generation of problem instances in Section IV-B. Numerical results are then presented in sections IV-C and IV-D.

A. Performance metrics

We consider two different performance metrics which were introduced in [12]. Given an instance of the problem, the *normalized number of queried edges* of SPD algorithm $\mathcal{A}$ on this instance is defined as

$$q_{\mathcal{A}} = \frac{|\mathcal{C}_{\mathcal{A}}|}{|\mathcal{E}|}, \tag{4}$$

and the *query ratio* of $\mathcal{A}$ on this instance is

$$r_{\mathcal{A}} = \frac{|\mathcal{C}_{\mathcal{A}}|}{|\mathcal{C}_{min}|}, \tag{5}$$

where $|\mathcal{C}_{min}|$ is the size of the smallest (in cardinality) certificate for the considered problem instance.

The normalized number of queried edges $q_{\mathcal{A}}$ is used to evaluate in which proportion the deployment time of a network slice can be reduced by querying the resource providers strategically, as compared to a strategy in which all edge costs would be collected at once. In contrast, the query ratio $r_{\mathcal{A}}$ measures how well algorithm $\mathcal{A}$ performs on a given instance, a value close to 1 indicating that the algorithm is almost optimal.

The evaluation of the query ratio implies to compute the minimum certificate $\mathcal{C}_{min}$. This can be done by solving the following Integer Linear Programming problem:

$$\min \sum_{e \in E} u_e \tag{6}$$

$$s.t.\ \delta^\star \leq \sum_{e \in \pi} u_e c_e + (1-u_e)c_e^{(0)},\ \forall \pi \in \mathcal{P}_{s,t} \tag{7}$$

$$u_e \in \{0,1\},\ \forall e \in \mathcal{E} \tag{8}$$

where u_e is the binary decision variable that indicates whether the edge $e \in E$ is uncovered or not, $\mathcal{P}_{s,t}$ is the set of all $s-t$ directed paths, and $\delta^\star$ denotes the length of a shortest path between s and t.

B. Generation of random instances

We shall evaluate the above performance metrics for the Greedy SPD algorithm over randomly generated problem instances. We consider symmetrical and asymmetrical network configurations, with different values of the parameters K, M and R, and generate random instances for each configuration. To generate these instances, we assume that the computational resources required in a Cloud to run a VNF are significantly less expensive than those required to interconnect VNFs, and similarly, that a virtual link within a Cloud is usually much cheaper than a virtual link between two geographically distant Clouds. More precisely, we assume that the cost of running a VNF is drawn from a uniform distribution in the set $\{1,\ldots,20\}$, whereas the cost of an intra-Cloud (resp. inter-Cloud) virtual link is uniformly distributed in the set $\{5,\ldots,30\}$ (resp. $\{10,\ldots,50\}$). We refer to the corresponding edges in the associated graph as type-1, type-2 and type-3 edges, respectively.

Asymmetrical configurations are obtained as follows. For each VNF, we choose randomly 2 (unless otherwise stated) out of the M Clouds as potential locations. Similarly, we choose randomly the number of network providers interconnecting two distant Clouds.

TABLE II: Distribution of the query ratio for 100 random instances without initial knowledge.

Query Ratio	Symmetric Configuration	Asymmetric Configuration
	Frequency	
[1 - 1.2]	14%	76%
[1.2 - 1.4[	76%	24%
[1.4 - 1.7[	10%	0%

C. Performance Metrics without Initial Knowledge

In this section, we assume that all edge costs are *totally unknown* ($c^{(0)} = 0$). We emphasize that, although the costs of the different types of edges are drawn from different random distributions, the algorithm is not aware of this and has the same initial knowledge for all edges.

We first consider the query ratio of the Greedy SPD algorithm in this case. As we need to solve ILP (6)-(8) for many random instances, we restrict ourselves to relatively small instances obtained for $K = 4$, $M = 5$ and $R = 5$. We report in Table II the distribution of the query ratios obtained over 100 random instances for symmetric and asymmetric configurations. For symmetric instances, the Greedy SPD algorithm is quite efficient as for 90% of the instances it only makes 40% more queries than the minimum required. The algorithm is even more efficient for asymmetric instances, since 3 out of 4 instances can be solved with only 20% more queries than the minimum required.

To evaluate the normalized number of queried edges, we consider larger instances obtained in two scenarios. In the first one, we set $M = 5$ and $R = 5$, whereas in the second one we set $M = 7$ and $R = 7$. Figure 6 shows the average value of the normalized number of queried edges as a function of K for both scenarios and for symmetric and asymmetric instances. Average values were computed over 500 random problem instances, and asymmetric instances were obtained by choosing randomly 4 out of the M clouds, and similarly 4 out of the R network providers. We observe from Figure 6 that the Greedy SPD algorithm achieves a sensible reduction of the

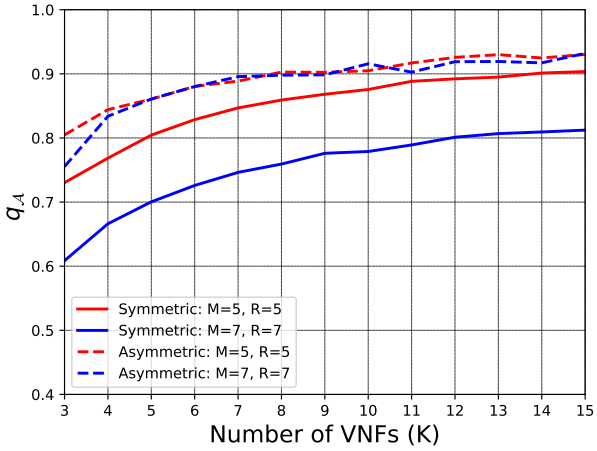

Fig. 6: Normalized number of queried edges as a function of K in the case of no initial knowledge.

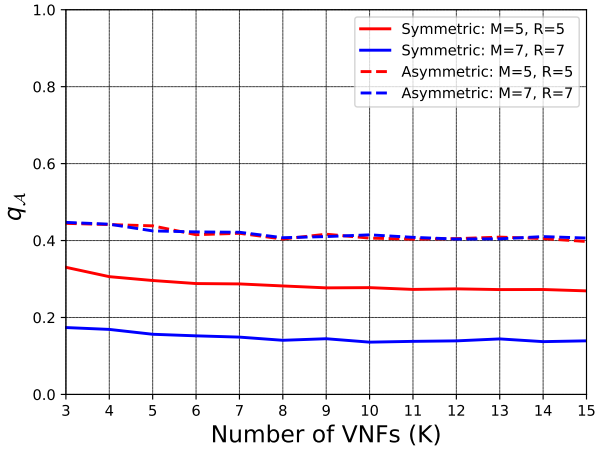

Fig. 7: Normalized number of queried edges as a function of K when some initial knowledge is provided.

TABLE III: Distribution of the query ratio for 100 random instances with initial knowledge.

Query Ratio	Symmetric Configuration	Asymmetric Configuration
	Frequency	
[1 - 1.2]	0%	21%
[1.2 - 1.4]	18%	57%
[1.4 - 1.6[	40%	20%
[1.6 - 1.8[	30%	1%
[1.8 - 2.4[	12%	1%

number of resource requests made by the slice provider. For instance, in the symmetric case, for $K = 5$ VNFs, the number of resource requests is reduced by 20% (resp. 30%) in the first (resp. second) scenario. For asymmetric instances, the gain is about 15% for both scenarios when $K = 5$. Note that these gains on the number of resource requests made by the slice provider directly translate into gains on the SFC deployment times.

D. Performance Metrics with some Initial Knowledge

We now assume some initial knowledge on the edge costs by setting the initial estimate $c_e^{(0)}$ to the lower bound of the interval used for randomly drawing the cost c_e of the edge (e.g., the estimate of a type-2 edge is set to 5). Note that the information provided to the algorithm is fairly modest as the costs of the edges are quite variable. We study the impact of this initial knowledge on the number of queries and on the query ratio.

The results obtained for the query ratio are shown in Table III. The setting considered is exactly the same as in Section IV-C. We observe a sensible degradation of the average query ratio, both for symmetric and asymmetric instances. This suggests that there is probably some room for improvement in the way the Greedy SPD algorithm handles the initial knowledge.

Nevertheless, the initial knowledge provided has a very notable impact on the number of queries made by the algorithm. Figure 7 shows the average value of the normalized number of queried edges as a function of K for the scenarios described in Section IV-C and for symmetric and asymmetric instances. The random instances are generated exactly as described in Section IV-C, the only difference being the initial information provided to the Greedy SPD algorithm. We observe drastic reductions in the number of resource requests made, and therefore in the SFC deployment time. For instance, in the symmetric case, for $K = 5$ VNFs, the number of resource requests is reduced by around 70% (resp. 85%) in the first (resp. second) scenario.

E. Performance Metrics with Different Edge Costs Distributions

The above numerical results suggest that a significant reduction of the number of resource requests made by the slice provider can be achieved. The order of magnitude of this reduction however clearly depends on the assumptions made for generating the edge costs. In this section, we evaluate the impact of the edge costs on the normalized number of queried edges. We generate random instances with the same settings as before, however we consider four distinct cost distribution setups:

1) *Setup-1:* Same as described in Section IV-B.
2) *Setup-2:* Continuous cost values. Similar to *Setup-1*, however the costs are drawn from a uniform distribution in a continuous set (e.g., the cost of running a VNF is drawn from the interval $[1, \ldots, 20]$).
3) *Setup-3:* No distinction between edges types. The costs of all the edges are drawn from a uniform distribution in the set $\{1, \ldots, 50\}$.
4) *Setup-4:* No intersection between the costs from different types of edges. More precisely, we assume that the cost of type-1 edges is drawn from a uniform distribution in the set $\{1, \ldots, 20\}$, whereas the cost of

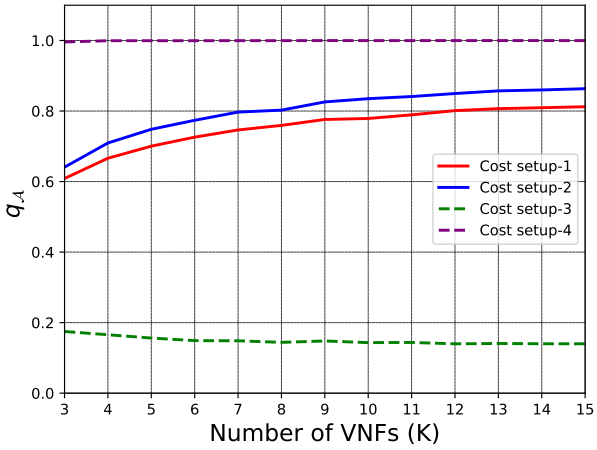

Fig. 8: Normalized number of queried edges as a function of K for a symmetric topology and no initial knowledge.

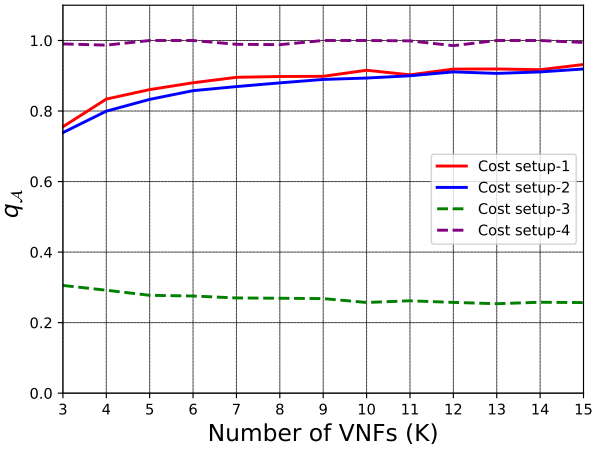

Fig. 9: Normalized number of queried edges as a function of K for an asymmetric topology and no initial knowledge.

type-2 (resp. type-3) edges is uniformly distributed in the set $\{21\ldots,30\}$ (resp. $\{31,\ldots,50\}$).

In figures 8, 9, 10 and 11, we report the normalized number of queried edges for different network topologies (symmetric and asymmetric) with different level of initial knowledge (none and some initial knowledge). We observe that the cost distribution setup has a major impact on the number of resource requests made. In particular, when the cost values of all edge types are drawn from the same set of values (cost *setup-3*), the reduction in the number of resource requests made is very high. For instance, in the symmetrical case without initial knowledge, the number of requests is reduced to below 20%. On the other hand, this number increases to almost 100% when the edge costs of the different types of edges are drawn from disjoint subsets(that is, *setup-4* is used) and there is no initial knowledge. This suggests that this a quite unfavourable case for the Greedy SPD algorithm, though we also surprisingly observe that in this case providing some initial knowledge enables to achieve the greatest gains. We observe less notable impact for non-integer cost values (*setup-2*) compared to *setup-1*.

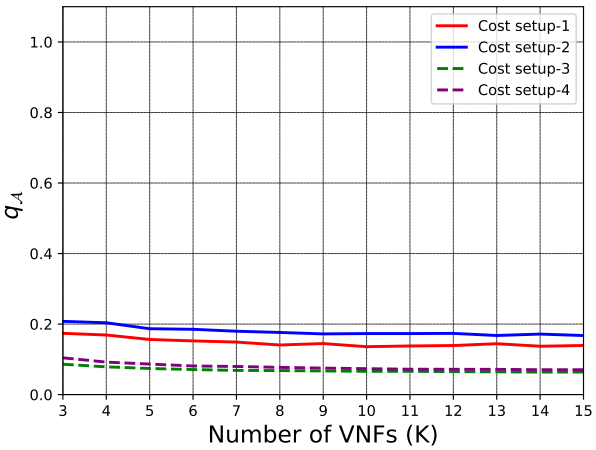

Fig. 10: Normalized number of queried edges as a function of K for a symmetric topology with some initial knowledge.

V. Conclusion

We have proposed a method based on a SPD algorithm for shortening the deployment time of SFCs in the SlaaS model. Our numerical results suggest that the deployment times could be reduced by $20-30\%$ by querying adaptively the resource providers instead of fetching all resource options at once, and even by $70-80\%$ when the slice provider is able to provide some initial estimates of allocation costs. The gains obtained are of course highly dependent on the assumptions and numerical values used, but we believe that the proposed approach can yield significant gains on deployment times in practice.

As future work, we plan to extend our model to the case of multiple source and destination nodes. Another interesting extension would be to consider the case where the slice provider does not necessarily require a minimum cost SFC embedding, but can be satisfied with an approximation of guaranteed quality.

References

[1] A. Gember, A. Akella, A. Anand, T. Benson, and R. Grandl, "Stratos: Virtual middleboxes as first-class entities," 2012.

[2] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, 2018.

[3] J. Elias, F. Martignon, S. Paris, and J. Wang, "Efficient orchestration mechanisms for congestion mitigation in NFV: Models and algorithms," *IEEE Transactions on Services Computing*, vol. 99, 2015.

[4] M. F. Bari, S. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *11th International Conference on Network and Service Management (CNSM)*, November 2015, pp. 50–56.

[5] N. Huin, B. Jaumard, and F. Giroire, "Optimal network service chain provisioning," *IEEE/ACM Transactions on Networking*, 2018.

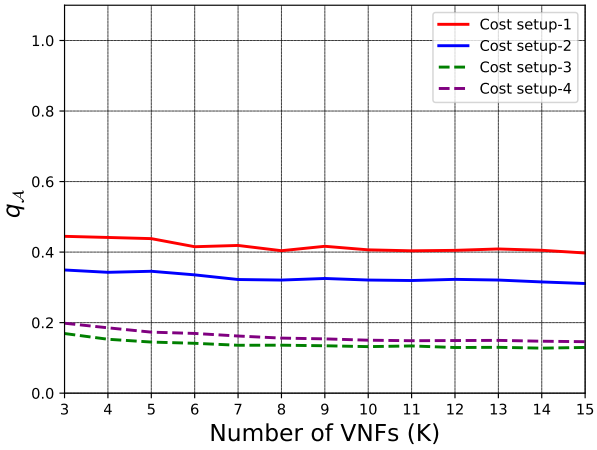

Fig. 11: Normalized number of queried edges as a function of K for an asymmetric topology with some initial knowledge.

[6] Z. Luo, C. Wu, Z. Li, and W. Zhou, "Scaling geo-distributed network function chains: A prediction and learning framework," *IEEE Journal on Selected Areas in Communications*, 2019.

[7] 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*, 2018.

[8] 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*, 2019.

[9] ETSI, "Management and orchestration; Provisioning," *TS 128.531, Release 15*, 2018.

[10] A. Dwaraki and T. Wolf, "Adaptive service-chain routing for virtual network functions in software-defined networks," in *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, New York, NY, USA, 2016, pp. 32–37.

[11] C. Szepesvári, "Shortest path discovery problems: A framework, algorithms and experimental results," in *AAAI*, 2004, pp. 550–555.

[12] C. Thraves Caro, J. Doncel, and O. Brun, "Optimal path discovery problem with homogeneous knowledge," *Theory of Computing Systems*, vol. 64, no. 2, pp. 227–250, 2020.