Algorithmes d'approximation pour le placement de chaînes de fonctions de services avec des contraintes d'ordre

Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, Stéphane Pérennes

Algorithmes d'approximation pour le placement de chaînes de fonctions de services avec des contraintes d'ordre[†]

Andrea Tomassilli[1] et Frédéric Giroire[1] et Nicolas Huin[2] et Stéphane Pérennes[1]

[1] *Université Côte d'Azur, I3S, CNRS, INRIA, COATI, Sophia Antipolis, France*
[2] *Université Concordia, Montréal, Canada*

Le modèle des réseaux programmables virtualisés permet aux opérateurs de télécommunications d'offrir des services réseaux complexes et flexibles. Un service se modélise alors comme une chaîne de fonctions réseaux (firewall, compression, contrôle parental,...) qui doivent être appliquées séquentiellement à un flot de données. Dans cet article, nous étudions le problème du placement de fonctions de services qui consiste à déterminer sur quels nœuds localiser les fonctions afin de satisfaire toutes les demandes de service, de façon à minimiser le coût de déploiement. Nous montrons que le problème peut être ramené à un problème de Set Cover, même dans le cas de séquences ordonnées de fonctions réseau. Cela nous permet de proposer deux algorithmes d'approximation à facteur logarithmique, ce qui est le meilleur facteur possible. Finalement, nous évaluons les performances de nos algorithmes par simulations. Nous montrons ainsi qu'en pratique, des solutions presque optimales peuvent être trouvées avec notre approche.

Mots-clefs : Virtualisation des fonctions réseaux, Chaînes de fonctions de service, Réseaux logiciels, Optimisation

1 Introduction

With the Network Function Virtualization (NFV) framework, network functions can be executed by generic hardware instead of dedicated equipment. Examples of network functions are firewall, load balancers, and Video Optimizer Controller. Network flows are often required to be processed by an ordered sequence of network functions. For instance, an Intrusion Detection System may need to inspect the packet before compression or encryption are performed. This notion is known as Service Function Chaining (SFC) [2].

The Problem. In this work, we consider the problem of placing service functions into generic hardware along the paths followed by flows. We assume a cost for installing a function on a node. The cost aims at reflecting the cost of having a virtual machine that runs a virtual function, such as license fees, network efficiency or energy consumption. The goal is to find the *placement of minimum cost*.

Contributions. We show that the problem can be reduced to a SET COVER Instance. This first shows that the problem is *NP*-Hard and hard to approximate within $\log(n)$, where n is the number of flows. Second, it allows us to provide two greedy algorithms with proven logarithmic approximation factor [3].

State of the art. The Placement problem has been widely studied. Existing placement algorithms can be roughly classified into two categories: ILP-based, lacking in scalability and greedy-based, with no provable performance guarantees. Closest works to ours are [4] and [5]. [4] addresses the problem of the placement of virtual functions within the physical network. The goal of the authors is to minimize the network cost. They provide near-optimal approximation algorithms with theoretically proven performance. However, the execution order of the network functions is not considered in their model. In [5], the authors focus their attention on the problem of optimal placement and allocation of VNFs to provide a service to all the flows of the network. The goal is to minimize the total number of network functions. In their model, flow routes

[†] A long version of this work has been accepted to INFOCOM 2018 [1].

are fixed, and one flow may be fractionally processed by the same network function at multiple nodes. However, they study the scenario of one single network function.

Content. We reduce the *SFC Placement Problem* to a standard *Set Cover* problem. This implies two algorithms. The fist one is based on the greedy technique and the second one on LP Rounding. Both algorithms achieve a logarithmic approximation ratio. Finally, we evaluate the performances of our proposed algorithms.

2 Problem Formulation

Given a digraph $G = (V, E)$, with a set of functions $\mathcal{F}$ and an installation cost c, the problem takes as an input a set of demands $\mathcal{D}$. The output consists of a minimum cost function placement that satisfies all the demands in $\mathcal{D}$. We refer to this problem as the *SFC Placement Problem*.

Input: A digraph $G = (V, E)$ and a collection $\mathcal{D}$ of demands. Each demand $d \in \mathcal{D}$ is associated with a path `path(d)` $\in V^*$ and to a sequence of functions `sfc(d)` $\in \mathcal{F}^*$. Lastly, a cost $c : V \times \mathcal{F} \to c(v, f)$, defining the cost of setting up the function f in node v.

Output: A *function placement* that is a subset $\Pi \subset V \times \mathcal{F}$ of function locations such that all demands of $\mathcal{D}$ are satisfied, i.e., for each demand the network functions appear in the correct order along the path.

Objective: minimize $\sum_{(v,f)\in\Pi} c(v, f)$

3 Approximation Algorithms for SFC-PLACEMENT

We show that the *SFC Placement Problem* is equivalent to an instance of the Minimum Weight Set Cover Problem. For each demand $d \in \mathcal{D}$, we denote with $l(d)$ and $s(d)$ the length of the associated path and chain respectively. Let `path(d)` $= u_1, u_2, ..., u_{l(d)}$ and assume that d requires the sequence of functions $sfc(P) = r_1, r_2, ..., r_{s(d)}$.

Given a demand d, we build a capacitated associated network $H(d, \Pi)$, as shown in Figure 1.

Definition 1. The network $H(d, \Pi)$ associated with a demand d is built as follows:

- $H(d, \Pi)$ has $s(d)$ layers $L_1, L_2, ..., L_{s(d)}$. Each layer contains $l(d)$ nodes corresponding to the nodes of `path(d)`. We note (u_i, j) the i-th node of layer j.
- There is an arc between the node (u, j) and the node $(v, j+1)$ if $u = v$ or if u precedes v in `path(d)`.
- $H(d, \Pi)$ has two other nodes, s_d and t_d. There is an arc between a node s_d and all the nodes of the first layer and an arc between all the nodes of the last layer and t_d.
- All arcs have infinite capacity, but each node has a capacity. The capacity of the node u of layer i is 1 if $(u, r_i) \in \Pi$ and 0 otherwise.

Lemma 1. *A demand $d \in \mathcal{D}$ is satisfied by Π if and only if there exists a feasible $st-path$ in the capacitated associated network $H(d, \Pi)$.*

With this notion of associated network, we define the following problem.

Problem 1. HITTING-CUT-PROBLEM $(\mathcal{D}, c)$ *is an instance of the Weighted Hitting Set problem where the elements are the function locations (u, f), for all $u \in V$ and $f \in \mathcal{F}$. Its cost is $c(u, f)$. The subsets of the universe correspond to all the st-vertex-cuts of the associated networks $H(d, \Pi)$ for all $d \in \mathcal{D}$.*

The problem is thus to find the sub-collection S of elements (functions placement) hitting all the subsets (cuts) of the universe of minimum cost.

Proposition 1. HITTING-CUT-PROBLEM $(\mathcal{D}, c)$ *is equivalent to* SFC-PLACEMENT $(\mathcal{D}, c)$.

Our problem is thus equivalent to a Hitting Set Problem, for which we know approximation algorithms. However, the number of st-vertex cuts is exponential in the number of vertices of the digraph. To derive a polynomial algorithm, we need to reduce the size of an instance of HITTING-CUT-PROBLEM. To this end, we use the fact that checking only the *extremal* cuts is enough (an *extremal* cut is a cut that is not strictly included in another cut) and that, in our problem, the extremal cuts of the associated graphs have a specific shape that we call *proper st-cuts*. See Figure 2 for an example.

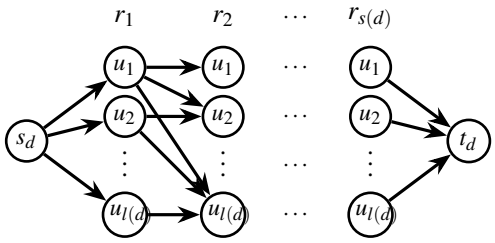

Fig. 1: The associated network of a demand $d \in \mathcal{D}$ routed on a path $\texttt{path(d)} = u_1, u_2, ..., u_{l(d)}$ that requires a chain $\texttt{sfc(d)} = r_1, r_2, ..., r_{s(d)}$

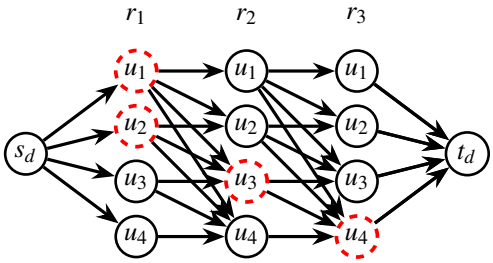

Fig. 2: Example of a proper cut (dashed nodes in red) for the layered graph relative to a demand d associated with a path of length 4 and a chain of length 3.

Definition 2. A proper st-cut of the associated graph $H(d,\Pi)$ is a cut of the following form:

$$\{\underbrace{(u_1,1),...,(u_{j_1},1)}_{\text{layer 1}},\underbrace{(u_{j_1+1},2),...,(u_{j_1+j_2},2)}_{\text{layer 2}},...,\underbrace{(u_{j_1+j_2+\cdots+j_{s(d)-1}+1},s(d)),...,(u_{l(d)=j_1+j_2+\cdots+j_{s(d)}},s(d))}_{\text{layer } s(d)}\}$$

for $j_1, j_2, ..., j_{s(d)} \geq 0$, such that $\sum_{i=1}^{s(d)} j_i = l(d)$.

Proposition 2. *The problem* SFC-PLACEMENT $(\mathcal{D},c)$ *is equivalent to a Hitting Set Problem with* $\sum_{d\in\mathcal{D}} \binom{l(d)+s(d)-1}{s(d)-1}$ *sets as an input. If each demand requires at most* $s_{\max}$ *network functions and is associated with a path of length smaller than* $l_{\max}$*, then the size of the instance is at most* $O(|\mathcal{D}| \cdot (l_{\max})^{s_{\max}-1})$.

3.1 A Greedy Algorithm

The main idea of the greedy algorithm is to avoid generating all proper cuts by showing it is enough to keep track of the *number of not hit proper cuts*. We show here that, by using dynamic programming, this number can be counted in time $O(|\mathcal{D}|l_{\max}^2 s_{\max})$.
For a demand $d = (\texttt{path(d)}, \texttt{sfc(d)})$, a function placement Π can be seen as a matrix A_d with $l(d)$ rows and $s(d)$ columns and for which $A_d[i,j] = 1$ iff $(u_i, r_j) \in \Pi$. We note $A_d[i:j, k:l]$ the submatrix of A_d considering only the rows from i to j and the columns from k to l. For a demand $d = (\texttt{path(d)}, \texttt{sfc(d)})$ and a function placement Π (or equivalently A_d), we note $N(d)$ the number of proper cuts not hit by A_d. It can be computed using the recursive function $N(r,c)$ defined below. We have $N(d) = N(l(d), s(d))$ with

$$N(r,c) = \mathbb{1}_{i^*(r,c)=0} + \sum_{j_c=0}^{r-i^*(r,c)} N(n-j_c, c-1), \text{if } c \geq 2$$
$$N(r,1) = \mathbb{1}_{i^*(r,c)=0}$$

where $i^*(r,c)$ is defined as follows. We consider the matrix $A_d[1:r, 1:c])$. We consider the ones placed in the last column of the matrix, column c. If there are none, $i^*(r,c) = 0$. Otherwise, $i^*(r,c)$ is the maximum index of such ones, that is, $i^*(r,c) = \max_{0\leq i\leq l(d)}\{i, \text{ such that } A_d[i,c] = 1\}$. $N(r,c)$ can be computed using dynamic programming. At each iteration, the algorithm selects the pair (u,f) with the smallest average cost per newly hit proper cut. The pair of minimum cost is added to the solution Π. Then, the number of remaining proper cuts to be hit is updated. This process is repeated until all the proper cuts are hit. The complexity of the whole procedure is $O(l_{\max}^2 s_{\max}|V|^2|\mathcal{F}|^2|\mathcal{D}|)$. The greedy algorithm achieves an approximation ratio equal to $\mathcal{H}(\#\text{Proper Cuts}) = \mathcal{H}(|\mathcal{D}|l_{\max}^{s_{\max}-1}) \sim \ln(|\mathcal{D}|) + (s_{\max}-1)\ln(l_{\max})$ [3], where $\mathcal{H}(n)$ is the n-th harmonic number.

3.2 LP Rounding Approach

The idea is to use the formulation of the problem looking for a path in the associated networks $H(d,\Pi)$. There are two kinds of binary decision variables.
(i) Location or capacity variables: $x(u,f)$ indicates whether the function f is installed on node u. It corresponds to the shared capacity of the node (u,f) of the associated networks.

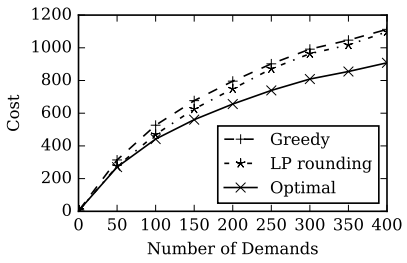

Fig. 3: Average setup cost as a function of the number of demands

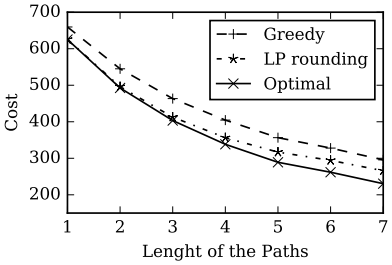

Fig. 4: Average setup cost as a function of the length of the paths

(ii) Flow variables. For each demand $d \in \mathcal{D}$, we have a flow variable f_{uv}^{d} for each edge of the associated network $H(d,\Pi)$. The constraints are (i) node capacity constraints and (ii) flow conservation constraints. Along with the greedy algorithm, we obtain a second approximation algorithm, with the same approximation factor. We refer to [1] for a detailed presentation of the model.

4 Numerical Results

In this section, we evaluate the performances of our proposed algorithms. We study how the total setup cost and the accuracy of our algorithms vary according to (i) different path lengths and (ii) increasing number of demands. We conduct experiments on a real-world topology: `germany50` (50 nodes and 88 links). We build our instances in the following way. The source and destination nodes of a demand are uniformly chosen at random from the set of vertices. The path of the demand is given by a shortest path between these two nodes and its chain is composed of 2 to 6 functions uniformly chosen at random from a set of 30 functions. Finally, the setup cost of a function on a node is uniformly chosen at random between 1 and 5.
In Figure 3, we compare the performances of the algorithms in the case of an increasing number of demands. The setup cost increases with the number of demands, as the number of functions to be placed increases. However, the increase is sublinear. The reason is that, the more demands in a network, the higher the opportunity of sharing functions. The optimality ratio is at most 21% for both algorithms.
In Figure 4, we only consider demands with pairs of nodes at equal distances, from 1 to 7. The total setup cost strictly decreases when the length of the path increases. In fact, when paths are longer, the demands tend (on average) to share more nodes, reducing the number of required functions to satisfy all the demands and so the cost. The ratio to the optimal solution never exceeds 25% for both algorithms.

5 Conclusion

In this paper, we investigated the problem of placing VNFs to satisfy the ordering constraints of the flows with the goal of minimizing the total setup cost. We proposed two algorithms that achieve a logarithmic approximation factor. Numerical results are given and validate the cost effectiveness of our algorithms.

References

[1] A. Tomassilli, F. Giroire, N. Huin, and S. Pérennes, "Provably efficient algorithms for placement of service function chains with ordering constraints," in *IEEE INFOCOM 2018*.

[2] P. Quinn and T. Nadeau, "Problem statement for service function chaining," 2015.

[3] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[4] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE INFOCOM 2015*.

[5] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *IEEE INFOCOM 2017*.