Virtual Network Function Deployment in Tree-structured Networks

Yang Chen, Jie Wu, and Bo Ji
Center for Networked Computing, Temple University, USA
Email: {yang.chen, jiewu, boji}@temple.edu

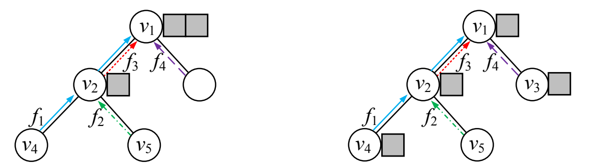
***Abstract*—Network Function Virtualization (NFV) evolves the implementation of network functions from expensive hardwares to software middleboxes. These software middleboxes, also called Virtual Network Functions (VNFs), are executed on switch-connected servers. Efficiently deploying such VNFs is challenging, because VNFs must fully process all flows with their traffic rates before they reach their destinations while VNF locations are restricted by the constraint of vertex capacity. In addition, each network function offers heterogeneous VNF types with different configurations of processing volumes and costs. This paper focuses on minimizing the total cost of deploying VNF instances for providing a specific network function to all flows in tree-structured networks. First we prove the NP-hardness of heterogeneous VNF deployment in a tree topology and propose a dynamic programming based solution with a pseudo-polynomial time complexity. Then we narrow down to three simplified cases by focusing on homogeneous VNFs or the linear line topology. Specifically, three algorithms are introduced: an improved dynamic programming based algorithm for deploying homogeneous VNFs in a tree topology, a performance-guaranteed algorithm for deploying heterogeneous VNFs in a linear line topology, and an optimal greedy algorithm for deploying homogeneous VNFs in a linear line topology. Extensive simulations are conducted to evaluate the performance of our algorithms.**

***Index Terms*—Deployment, NFV, SDN, tree-structured networks, VNFs.**

I. Introduction

Network Function Virtualization (NFV) addresses the problems of traditional purpose-built hardware appliances [1] by leveraging virtualization technologies to implement network functions in software [2] such as firewalls, network address translator, proxies, and deep packet inspection. Software middleboxes, also called Virtual Network Functions (VNFs) [3], are provisioned most commonly in modern networks to demonstrate their increasing importance [4]. With the emergence of Software Defined Networking (SDN), there is a tendency to incorporate SDN and NFV in concerted ecosystems [5]. SDN maneuvres traffic through appropriate VNFs and allows VNFs to pick service locations from multiple available servers; on the other hand, traditional hardwares leave no choice for allocations [6]. This results in a flexible architecture and has the potential to significantly reduce capital and operating expenses, shorten product release cycle, and improve service agility.

This paper studies the VNF deployment problem with a given set of flows in tree-structured networks, whose switch-connected servers have limited capacities (the maximum number of deployed VNF instances). Tree-structured topologies are quite common in streaming services and Content Delivery Networks (CDNs) [7]. Additionally, it is proven NP-hard to minimize the total number of VNF instances even to deploy one service function in a general topology [10]. Thus, we narrow down to tree-structured networks and provide stronger algorithmic results in this paper. We assume that all flows are upstream (destination is closer to the root than source) and require an identical network function, which has heterogeneous VNF types with different configurations of processing volumes and costs [11]. The processing volume of a VNF instance can be shared by multiple flows. A flow can be fractionally processed by several instances before its destination [10]. Our objective is to minimize the total deployment cost when all flows are fully processed with their traffic rates before reaching destinations.



(a) Unlimited vertex capacity. (b) Limited vertex capacity of one instance.

Fig. 1: A motivating example.

However, most existing works assume that the vertex capacity is unlimited or the number of instances is much smaller than the vertex capacity. We use an example in Fig. 1 to illustrate the complexity of the VNF deployment problem without and with the limited server capacity constraint. The topology of the toy example is a binary tree with five vertices. There are four flows, f_1, f_2, f_3, and f_4, whose sources, destinations, and paths are shown in Fig. 1. Their traffic rates are $3, 3, 4$, and 2, respectively. We are given a single type of VNF instance m (grey square box) with a processing volume 4 and a cost 1. We aim at minimizing the total cost of deploying m when the traffic rates of all flows are fully processed before destinations. Fig. 1(a) shows the optimal deployment with unlimited vertex capacities by applying the algorithm in [10]. The full traffic rate of f_1 and 1 traffic rate of f_2 are processed by the deployed instances on v_2, while the rest rates are processed by the two instances deployed on v_1. The total cost is 3 because of deploying 3 instances. As for the limited server capacity case, if each server can place at most one instance, one optimal deployment with a minimum cost of 4 is shown in Fig. 1(b).

Compared to Fig. 1(a), one more instance is deployed since v_1 can deploy only one instance. In order to fully process all flows before destinations, both instances on v_1 and v_4 waste 1 processing volume while the one on v_3 wastes 2. The waste is unavoidable because of the vertex capacity limitation and the service requirement.

The main challenges of our deployment problem lie in the selection of VNF locations and the allocation of each deployed VNF processing volume. The vertex capacity constraint complicates the deployment, since flows have to be fully processed before reaching their destinations. Intuitively, if we deploy the instances too close to the root of the tree, the processing volume is more likely to be used up, while flows with destinations far from the root may not be processed; if too far from the root, the opportunity of sharing the processing volume of an instance is scarce so that some will be wasted and more VNFs are needed. Additionally, heterogeneous VNF types of configurations for a network function, which have not been studied in the deployment problem, offer more deployment options and make the problem more complex.

In this paper, we first solve the heterogeneous VNF deployment problem in a tree topology with a dynamic programming based method. Because of NP-hardness of the problem, the solution is pseudo-polynomial and its time complexity is not easily tractable. Then we study a special case of homogeneous VNF deployment and improve the dynamic programming solution with an acceptable time complexity. Additionally, the heterogeneous VNF deployment problem in a linear line topology can be transformed to the classic submodular set cover problem so that we introduce a performance-guaranteed greedy strategy. An optimal greedy algorithm is designed for the simple case: homogeneous VNF deployment in a linear line topology.

Our main contributions are summarized as follows:

- We prove the NP-hardness of the heterogeneous VNF deployment problem in the tree-structured network.
- We propose four pseudo-polynomial algorithms in different settings of topologies and VNF types of configurations, shown in Tab. I with properties and time complexities[1]. Since $|M|$ (total number of VNF types of configurations) and c_{max} (largest vertex capacity) are small and integer-valued, while w_{max} (largest single VNF instance setup cost) is in an arbitrary precision and order of magnitude, the first algorithm is computationally hard, and the complexities of the rest of the three algorithms are dramatically improved.
- Extensive simulations are conducted to evaluate the efficiency of our proposed algorithms.

The remainder of this paper is organized as follows. Section II surveys related works. Section III describes the model, formulates the problem, and shows hardness. Section IV introduces our deployment algorithms in tree-structured topologies. In Section V, we handle cases in line topologies. Section VI includes the experiments, and Section VII concludes the paper.

[1]An algorithm has pseudo-polynomial time if its running time is a polynomial in the numeric value of the input (the largest integer present in the input) (e.g., c_{max} in Tab. I), instead of the length of the input (the number of bits required to represent it) (e.g., V in Tab. I), which is the case for polynomial time algorithms.

TABLE I: Our proposed solutions and time complexities.

Topo \ Type	Heterogeneous		Homogeneous	
Tree	DP	Optimal	DP	Optimal
		$O(\|V\|^4 \times (c_{max} \times w_{max})^3)$		$O(\|V\|^4 \times (c_{max})^3)$
Line	Greedy	Approximate	Greedy	Optimal
		$O(\|V\|^2 \times \|M\| \times c_{max})$		$O(\|V\| \times c_{max})$

II. Related Work

NFV frameworks have drawn a lot of attention, especially in the area of VNF deployment problem. Various objectives with different backgrounds are conduced in recent years. In this section, we give a brief review of state-of-art works.

Casado et al. [12] propose a model for deploying a single type of VNFs and present a heuristic algorithm to solve the deployment problem. [10] studies the joint deployment and allocation of a single type of VNFs, where flows can be split and fractionally served by several VNF instances. They propose several performance-guaranteed algorithms to minimize the number of VNF instances. However, they treat all servers with unlimited capacities such that they are able to hold an arbitrary number of VNF instances, which is not practical. [13] is the first to study the VNF deployment problems taking the effects of changing traffic volume into consideration. It also studies the multiple VNF deployment of different dependency relationships. They target load balancing through VNF deployment and flow-routing path selection. However, this work only processes a single flow and takes no consideration of the limited VNF processing volume. It results in exclusive instances for each flow, which is wasteful of server resources.

There are other types of service coverage for each flow, such as service chain where each flow has to be covered by a sequence of services with or without particular order, instead of single service used in our model. Rost et al. [14] prove the NP-completeness and inapproximability of the service chain deployment under different constraint settings, extended from the virtual network embedding problem. They initiate the study of approximation algorithms and propose a performance-guaranteed solution under the offline setting (given multiple flows), based on randomized rounding of Linear Programming, to maximize the total profit of satisfied flows in [15]. Since our model is special with one service in a service chain, the results obtained in this paper are more specific. In tree-structured networks, we propose optimal DP-based solutions of the VNF deployment.

III. Model and Formulation

A. Network Model

We first present our model of the directed tree-structured network, $T = (V, E)$, where $V = \{v\}$ is a set of vertices (i.e., switches), and $E = \{e\}$ is a set of directed edges (i.e., links).

We use v to denote a single vertex and vertices are labeled of $1, 2, ..., |V|$ by the Breadth-First-Search (BFS). (We use $|\cdot|$ to denote the cardinality of a set.) Each vertex v_i is connected to a capacity-limited server. The vertex capacity, denoted as c_i, represents the maximum number of VNF instances that can be deployed on v_i. For each location on v_i, we can deploy one VNF instance with any type of configuration. We use two definitions of tree data structure to simplify our discussion.

***Definition** 1 (height, subtree):* The *height* of a vertex is 1 plus the difference between the depth of the tree and the depth of the vertex. A *subtree* T_i of a vertex v_i in a tree T is a tree consisting of v_i and all its descendants in T.

Take the tree in Fig. 1(a) as an example. The height of v_1 is 3 and the heights of v_3 and v_4 are 2 and 1, respectively. The subtree T_2 consists of v_2, v_4 and v_5.

We are given a set of flows $F = \{f\}$ and all flows request to be processed by an identical network function (service). All flows are upstream flows, i.e. the source of a flow is a descendant of its destination. We use f to denote a flow with a source of src_f, a destination of dst_f, and an initial traffic rate of r_f. We say that a flow is satisfied when its initial traffic rate is fully processed before reaching its destination.

$M = \{m\}$ is the set of VNF types with different configurations for the requested network function (service). Each VNF type m has a processing volume, α_m, which is the maximum total traffic rate that one m instance can process, and a setup cost w_m for setting up one VNF instance of type m. Different VNF types of configurations for a network function provide the same network service, but have various processing volumes and setup costs. We simplify the types of different configurations as different types in the following.

***Definition** 2 (heterogeneous, homogeneous):* VNFs are called *heterogeneous* if the number of VNF types is more than one; otherwise, they are called *homogeneous*.

We assume each flow can be fractionally processed by several VNF instances of any type deployed on vertices along its path. We introduce the definition of the deployment plan and its feasibility.

***Definition** 3 (deployment plan, feasibility):* A *deployment plan of* v, denoted as Ω_v, is a set of VNF instances with different types deployed on v. These instances are labeled by $1, 2, ..., |\Omega_v|$. A *deployment plan of the tree* T, denoted as Ω, is the union set of Ω_v, $\forall v \in V$, i.e. $\Omega = \{\Omega_v | v \in V\}$. We call a deployment plan *feasible* when all flows are fully processed before destinations.

Note that we can check the existence of a feasible deployment plan by deploying all the VNF instances with the maximum processing volume in all available locations of servers. If this deployment plan is still not feasible, then no feasible deployment exists.

We use $m(v, j) \in \Omega_v$ to record the j_{th} placed VNF instance on v after the labeling. The processing volume and setup cost of $m(v, j)$ are expressed as $\alpha(v, j)$, and $w(v, j)$, respectively. Let $\lambda^f_{m(v,j)}$ denote the amount of f's traffic rate processed by the j_{th} VNF instance deployed on v. Here each packet of flows should only be processed by instances once, because being processed by any instance will add an extra transmission delay, which should be avoided. In the following, we use the superscript max to denote the maximum value in a set such as $w_{max} = \max_{m \in M} w_m$ and $c_{max} = \max_{v \in V} c_v$. For the ease of reference, we summarize notations in Tab. II.

TABLE II: Symbols and Definitions.

Symbols	**Definitions**
V, E, F, M	the set of vertices, edges, flows, and VNF types
v, f, Ω	a vertex, a flow, and a deployment plan
Ω_v, c_v	the deployment plan and vertex capacity of v
src_f, dst_f, r_f	source, destination, initial traffic rate of f
$m(v, j)$	the j_{th} placed instance on v
$t(v, j), \alpha(v, j), w(v, j)$	type, volume and cost of j_{th} instance on v
$\lambda^f_{m(v,j)}$	traffic rate of f processed by $m(v, j)$

B. Problem Formulation

In this paper, we study the VNF deployment problem: given a set of flows F in a tree-structured network T, we deploy heterogeneous VNFs with the minimum total cost to satisfy all requests of flows.

***Definition** 4 (total cost):* The total cost of a deployment plan Ω is the summed-up cost of setting up all VNF instances, denoted by $cost(\Omega)$, which satisfies $cost(\Omega) = \sum_{v \in V} \sum_{m(v,j) \in \Omega_v} w(v, j)$.

Our problem can be formulated as:

$$\min \ cost(\Omega) \qquad (1)$$

$$\text{s.t. } |\Omega_v| \leq c_v \qquad \forall v \in V \qquad (2)$$

$$\sum_{v \in V} \sum_{j} \lambda^f_{m(v,j)} \geq r_f \qquad \forall f \in F \qquad (3)$$

$$\sum_{f \in F} \lambda^f_{m(v,j)} \leq \alpha(v, j) \qquad \forall m(v, j) \in \Omega_v, v \in V \qquad (4)$$

Our objective is to minimize the total cost of deployed VNF instances in Eq. (1). Eq. (2) states that the total number of deployed instances of each vertex is within its capacity. Eq. (3) guarantees each flow being fully processed by its initial traffic rate. Eq. (4) requires that the sum of processed traffic rate by each VNF instance is no more than its processing volume on all vertices.

C. Problem Hardness Analysis

In a general topology with homogeneous VNF instances, [10] proves that it is NP-hard to minimize the total deployed instance number, which is equivalent to minimizing the total cost of the deployment. Here we study the hardness of deploying the heterogeneous VNFs and we have:

***Theorem** 1:* The *heterogeneous VNF deployment* with the minimum cost is NP-hard even in a line topology.

The proof can be found in Appendix A. It is worth mentioning that we can apply the PTAS solutions in [16] if all flows have the same paths, i.e., the topology is a line. However, whether there exist PTAS solutions for the case with general topologies remains an open question.

IV. VNF Deployment in a Tree Topology

This section studies the deployments of heterogeneous and homogeneous VNFs in tree-structured topologies.

A. Heterogeneous VNF deployment in a tree topology

First we handle the most general case. We propose a dynamic programming based solution of the heterogeneous VNF deployment problem in a tree topology, called Heterogeneous Dynamic Programming algorithm (HeteDP).

Before the recurrence, we define some notations. Let $OPT(i,w)$ denote the minimum total unprocessed rate going out of node v_i by deploying VNFs with a total cost w in the subtree of v_i. If we are unable to fully process flows having $dst_f \in T_i$ by a total cost w, we have $OPT(i,w) = \infty$. This is because the destination is the last chance of a flow to be processed. We prioritize processing flows with smaller-height destinations since their opportunities of being processed are less. We use $w(l)$ and $w(r)$ to denote the allocated costs of v_i's left and right subtrees, respectively. $Deploy(i,w)$ denotes the maximum total processing volume by deploying instances with a total cost w on v_i. The relation of the minimum total unprocessed traffic rates out of v_i and its children can be formulated as:

$$OPT(i,w) = \max\{0, \min_{\substack{w(l)+w(r)\leq w \\ w(l),w(r)\geq 0}} \{\sum_{src_f=v_i} r_f + OPT(2i, w(l)) + OPT(2i+1, w(r)) - Deploy(i, w-w(l)-w(r))\}\} \quad (5)$$

Eq. (5) states that $OPT(i,w)$ equals 0 if there is a deployment plan able to process all unprocessed rates by deploying instances with a total cost w in the subtree of v_i; otherwise, it equals the minimum total unprocessed traffic rate out of node v_i. We combine all possible allocations of the total cost w among v_i's children and itself by changing $w(l)$ and $w(r)$.

To prove its optimality, let's consider one of the optimal deployments as Ω^* when given a VNF deployment problem. Here are some observations of Ω^*: (i) If Ω^* deploys instances with a fixed total cost w in the subtree of a vertex v, it should process as much traffic rate as possible. In other words, the total unprocessed traffic rate going out of v (upwards to its parent) should be minimized with the allocated cost w. This is because the more unprocessed traffic rate is out of v, the larger cost the deployment of v's ancestors is likely to have. (ii) The unprocessed traffic rate passing through v comes from two kinds of flows: flows with $src_f = v$ (flows start at v) and flows with some unprocessed traffic rate and $src_f \in T_v \setminus v$ (not-fully-processed flows coming up from its subtrees). (iii) The total deployment costs of all subtrees of v's children must be no more than w. Suppose each child vertex v_i deploys instances with a total cost w_i in the optimal deployment Ω^*, then instances with a total cost $w - \sum_{v_i \in T_v} w_i \geq 0$ will be deployed on vertex v. (iv) With a fixed value of w_i for the subtree of v_i, its deployment plan should also have the minimum total unprocessed traffic rate going upwards out of v_i in order to lower the potential cost of deployed instances of v_i's ancestors. (v) As the optimal deployment should have the minimized unprocessed traffic rate going out of v, the deployed instances on v with a total cost $w - \sum_{v_i \in T_v} w_i$ should have the maximum total processing volume.

Algorithm 1 Heterogeneous DP (HeteDP)

In: Sets of vertices V, edges E, flows F, VNFs M;
Out: The minimum total cost of deployed VNFs and the deployment plan Ω;

Initiate the array of OPT;
Generate the array of $Deploy$;
for each node v_i from bottom-up **do**
    for $w \in [0, \sum_{v \in T_i} c_v \times w_{max}]$ **do**
        Use the recurrence Eq. (5) to compute $OPT(i,w)$;
        if $OPT(i,w) = 0$ **then** break;
Find Ω with the minimum w making $OPT(1,w) = 0$;
return The deployment plan Ω.

With the insights above, our objective of the deployment problem is equivalent to finding the minimum cost of making the unprocessed traffic rate out of v_1 as low as 0. Moreover, the optimal deployment of a tree T with the root v_1 is able to be separated into a polynomial number of subproblems in its children. The optimal solutions of its children with different allocated cost combinations yield an optimal deployment to v_1, and we can build up solutions to these subproblems using a recurrence. It is worth mentioning that there are exponential combinations of the costs that are allocated to the vertex itself and all subtrees of its children when the total cost is fixed. In order to generate the optimal deployment plan, we need to list all such combinations, which is exponential of the number of v's children and w. In this paper, we only discuss the binary tree topology to reduce the number of combinations to polynomial of w. As a result, we can generate an optimal solution with an acceptable time complexity.

As for the item $Deploy(i, w-w(l)-w(r))$ in Eq. (5), we should maximize it in order to minimize the total unprocessed traffic rate out of v_i. It means to process the maximum total traffic rate by deploying VNF instances on v_i with a cost of $(w-w(l)-w(r))$, which can be formulated as following:

$$\max \sum_{m(i,j)\in\Omega_i} \alpha(i,j) \quad (6)$$

$$s.t. \sum_{m(i,j)\in\Omega_i} w_i(j) \leq w - w(l) - w(r) \quad (7)$$

$$|\Omega_i| \leq c_i \quad (8)$$

The formulation is the same as the classic knapsack problem [17] except the second constraint. In the knapsack problem, we are given a set of items, each of which has a non-negative weight and a distinct benefit. We need to find a subset with the maximum total benefit subject to the constraints that the total weight of the subset should not exceed specific values. The processing volume α_m and the setup cost w_m correspond to the benefit and weight in the knapsack problem, respectively. We slightly modify the dynamic programming solution of the knapsack problem proposed in [18]. We use $vol(w)$ to denote the maximum total processing volume that can be attained with a total deployment cost no more than w. The value of $vol(w-w(l)-w(r))$ is the solution to our problem. Suppose $vol(0)=0$, then the recurrence can be justified as $vol(w)=\max_{m\in M}\{\alpha_m+vol(w-w_m)\}$. When the number of selected items reaches c_v, the total processing volume $vol(w)$ keeps unchanged by not adding more items even when the weight w is not used up. This is because we need to control not only the total cost less than $w-w(l)-w(r)$, but also the number of selected items less than the vertex capacity. Thus, we list all combinations of possible deployments on v_i and find the feasible one with the largest processing volume as Ω_v.

***Lemma** 1:* The worst time complexity of generating the $Deploy$ array is $O((c_{max})^2\times w_{max})$.

Proof: The modified knapsack problem can be solved in $O(c_v\times(w-w(l)-w(r)))$ time complexity. We find that the solution of our modified knapsack problem is independent of the deployment plan. In order to lower the time complexity of HeteDP, we can calculate the *Deploy* array in advance and refer to its values when applying the HeteDP algorithm. The worst time complexity of the modified knapsack problem is $O(c_{max}\times(c_{max}\times w_{max}))=O((c_{max})^2\times w_{max})$. This is because the maximum deployment on a vertex is to deploy the most expensive VNF on all available locations. ■

We propose the HeteDP algorithm in Alg. 1. We initiate all value of $OPT(i,w)$ as 0 in line 1. We calculate the recurrence in Eq. (5) for each vertex from bottom-up in lines 3-5. Whenever $OPT(i,w)=0$, we break the current loop and continue to do the next loop in line 6. We find the minimum w making $OPT(1,w)=0$ in line 7 and return the corresponding deployment plan Ω by tracing back in line 8. We analyze the time complexity of our algorithm as follows.

***Theorem** 2:* The worst time complexity of HeteDP algorithm is $O(|V|^4\times(c_{max}\times w_{max})^3)$.

Proof: HeteDP algorithm is a pseudo-polynomial time algorithm using dynamic programming method. First, all $|V|$ vertices need to be traversed so that the algorithm has $|V|$ iterations. Second, in each iteration of a vertex from bottom-up, we try all possibilities of the cost value w. The maximum cost value is $O(\sum_{v\in V}c_v\times w_{max})=O(|V|\times c_{max}\times w_{max})$. Next for a fixed cost value w for the subtree of a vertex v, we need to list all combinations of allocating the cost w to itself and its two children while ensuring $w(l)+w(r)\le w$. There are at most $O((|V|\times c_{max}\times w_{max})^2)$ combinations. Then for each combination, we need a constant time to calculate the value of $\sum_{src_f=v_i}r_f+OPT(2i,w(l))+OPT(2i+1,w(r))-Deploy(i,w-w(l)-w(r))$ by referring to the OPT array as well as the $Deploy$ array. As discussed in Lemma 1, the generation of all values in the $Deploy$ array takes at most $O((c_{max})^2\times w_{max})$ time and we only need to calculate it once. We determine the minimum value by traversing the values of all combinations in a $O((|V|\times c_{max}\times w_{max})^2)$ time and calculate the value of $OPT(i,w)$. Finally, the worst time complexity is the number of iterations, times the maximum number of cost value, times the maximum number of combinations of a fixed cost value, which is $O(|V|\times(|V|\times c_{max}\times w_{max})\times(|V|\times c_{max}\times w_{max})^2)=O(|V|^4\times(c_{max}\times w_{max})^3)$. ■

Note that we can also lower the time complexity by stopping increasing w of $OPT(i,w)$ in two cases: the first case is when the smallest w for the vertex v_i appears making $OPT(i,w)=0$; the second case is when w reaches $\sum_{v\in T_i}c_v\times w_{max}$. This is because $OPT(i,w)=0$ means that there is no unprocessed traffic rate out of v_i, meaning that instances with a cost w can process all flows in the subtree of v_i. A larger w is unable to process any more flows, since no unprocessed flow exists. In addition, finding the minimum value of w making $OPT(1,w)=0$ is our objective. The second case states the natural upper bound of w that all available locations in the subtree of T_i are deployed by the most expensive instance.

***Theorem** 3:* HeteDP is optimal for heterogeneous VNF deployment in a tree topology.

The detailed proof is omitted due to the optimal property of the dynamic programming method.

B. Homogeneous VNF deployment in a tree topology

First we present a lemma to transform our objective into a simpler equivalent form when there is only one type of VNFs.

***Theorem** 4:* Minimizing the total cost of deployed instances with homogeneous VNFs is equivalent to deploying the minimum number of instances.

Proof: As there is only a single type of VNF m, our cost function can be converted to $cost(\Omega)=\sum_{v\in V}|\Omega_v|\times w_m=|\Omega|\times w_m$. Since w_m is a constant, it is the same as minimizing $|\Omega|$, which is the total number of deployed instances. ■

Our objective is transformed to minimizing the total number of deployed VNF instances when there is only a single type of VNF m. Inspired by HeteDP, we also propose a dynamic programming based algorithm, called HomoDP, which is simpler and more tractable than HeteDP. We replace the total cost w by the total number of deployed instances n in each subtree of vertices. We use $OPT(i,n)$ to denote the minimum total unprocessed traffic rate going out of node v_i by deploying n VNF instances altogether in the subtree of node v_i. Our target is to find the minimum n making $OPT(1,n)=0$. If flows with destinations within the subtree of v_i are unable to be fully processed by deploying n instances, we have $OPT(i,n)=\infty$. We also prioritize processing flows with smaller-height destinations. We use $n(l)$ and $n(r)$ to denote the deployed instances in v_i's left and right subtrees, respectively. There are $n-n(l)-n(r)$ VNF instances to be deployed on v_i. We replace the $Deploy(i,w-w(l)-w(r))$ by $(n-n(l)-n(r))\times\alpha_m$. HomoDP's similar recursive formulation is omitted because of limited space.

Here we use the topology in Fig. 1(b) with the same setting as an example to show the deployment procedure. The tree has five nodes with capacities $c_v=1,\forall v\in V$. There are four flows f_1,f_2,f_3 and f_4 with initial traffic rates as $r_1=3,r_2=3,r_3=4$, and $r_4=2$. There is only one type of VNF m with $\alpha_m=4$. We aim to find the smallest n such that $OPT(1,n)=0$. For ease of reference, we list the values of $OPT(i,j)$ in Table III. We traverse

TABLE III: The values of $OPT(i,n)$.

i \ n	0	1	2	3	4
1	∞	∞	∞	∞	0
2	∞	6	2	0	0
3	2	0	0	0	0
4	3	0	0	0	0
5	3	0	0	0	0

vertices from bottom-up by first calculating $OPT(5,0) = r_2 - 0 = 4$. We have $OPT(5,1) = \max\{0, r_2 - 1\times\alpha_m\} = \max\{0, 3-4\} = 0$. As $c_5 = 1$, more than one instance are unable to be deployed resulting in $OPT(5,n) = 0$, $\forall n{\geq}2$. Similarly, we can calculate $OPT(3,n)$ and $OPT(4,n)$, $\forall 0{\leq} n \leq 4$. Since f_2 with $dst_2 = v_2$ is not processed by not deploying any VNF in the subtree of v_2 ($n = 0$), we have $OPT(2,0) = \infty$ indicating the infeasibility of the deployment. The detailed calculation of $OPT(2,1)$ is that $OPT(2,1) = \max\{0, \min\{r_3+OPT(4,1)+OPT(5,0)-0\times\alpha_m, r_3+OPT(4,0)+OPT(5,1)-0\times\alpha_m, r_3+OPT(4,0)+OPT(5,0)-1\times\alpha_m\}\} = \max\{0, \min\{4+3+0-0, 4+3+0-0, 4+3+3-4\} = 6$. Similarly, we calculate other values of OPT array in Tab. III. The smallest n making $OPT(1,n) = 0$ is 4. By tracing back the table, the optimal deployment Ω is as shown in Fig. 1(b).

***Theorem** 5:* The worst time complexity of the HomoDP algorithm is $O(|V|^4 \times (c_{max})^3)$.

Proof: As HomoDP is simplified from HeteDP when w_{max} is a constant, then the complexity in Theorem 2 is reduced to $O(|V|^4 \times (c_{max})^3)$. ■

***Theorem** 6:* HomoDP is optimal for homogeneous VNF deployment in a tree topology.

The detailed proof is omitted due to the optimal property of the dynamic programming method.

V. VNF Deployment in a Line Topology

In this section, we simplify the tree-structured topologies into lines in order to generate more efficient algorithms.

A. Heterogeneous VNF deployment in a line topology

In this subsection, we simplify the tree topology into a line and propose a performance-guaranteed algorithm of deploying heterogeneous VNFs. We are given a line topology $L = (V, E)$ with $|V|$ nodes (vertices), which are labeled $1, 2, ..., |V|$ by a line coordinate axis. For simplicity, we say that one vertex is *smaller (larger)* than another vertex if its coordinate is smaller (larger) and vice versa. Assume the source of each flow is smaller than its destination no matter where its source and destination reside in the line. This means that flows transfer from left to right. When deploying one new instance of type m on v, we omit the sequence number of the j_{th} instance $m(v,j)$ by denoting the instance as $m(v)$. The new deployment plan is expressed as $\Omega \cup m(v)$. Before proposing our solution, we introduce two definitions.

***Definition** 5 (benefit function):* The benefit function, denoted as $b(\Omega)$, indicates the total processed traffic rate of a deployment plan Ω, which satisfies $b(\Omega) = \sum_{v\in V}\sum_{m(v,j)\in\Omega_v}\sum_{f\in F}\lambda^f_{m(v,j)}$.

***Definition** 6 (marginal benefit):* The marginal benefit, denoted as $b_\Omega(m(v)) = b(\Omega\cup m(v)) - b(\Omega)$, indicates the marginal contribution of processing flows by deploying a new instance of type m on v beyond the current deployment Ω.

We analyze the property of the benefit function $b(\Omega)$. A function f is *submodular* if and only if $\forall S \subseteq T \subseteq N, \forall e \in N\setminus T$, $f_T(e) \leq f_S(e)$. Then we prove that $\forall m(v) \notin \Omega'$, if $\Omega \subseteq \Omega'$, the submodular property holds, i.e., $b(\Omega\cup m(v)) - b(\Omega) \geq b(\Omega'\cup m(v)) - b(\Omega')$.

***Theorem** 7:* $b(\Omega)$ is a submodular function.

Proof: $b(\Omega)$ is an non-decreasing function, which is monotone. Suppose two deployment Ω and Ω' with $\Omega \subseteq \Omega'$. It is intuitive that the more instances are selected, the less unprocessed traffic rates remain, since the newly added m can only process the unprocessed rate. The maximum marginal benefit of a VNF instance m is α_m because of its processing volume limitation. If the newly added instance processes no traffic rate in both Ω and Ω', then $b(\Omega\cup m(v)) - b(\Omega) = b(\Omega'\cup m(v)) - b(\Omega') = 0$. As long as m process some flows in Ω', it will process no less traffic rate in Ω. Then we have $b(\Omega\cup m(v)) - b(\Omega) \geq b(\Omega'\cup m(v)) - b(\Omega')$. Thus, $b(\Omega)$ is a submodular function. ■

Here we explain that our problem formulation in Section III(B) can be transformed to the classic submodular set cover problem [19]. Our objective $cost(\Omega)$ in Eq. (1) is an non-decreasing function. The two constraints in Eqs. (2) and (4) are included in the definition of our $b(\Omega)$ function. Specifically, the ground set of the benefit function $b(\Omega)$ limits the available deploying locations within each vertex's capacity, and the marginal benefit limits the largest contribution of an instance no more than its processing volume. $b(\Omega)$ is the non-decreasing, submodular set function proved in Theorem 7. The constraint in Eq. (3) corresponds to the covering requirement of the set cover problem that each flow needs to be fully processed. Then our problem can be transformed as:

$$\min \quad cost(\Omega) \tag{9}$$

$$s.t. \quad b(\Omega) \geq \sum_{f\in F} r_f \tag{10}$$

Before introducing the solution, we sort flows in an alphabetical order of a tuple $< dst_f, src_f >$ (the ascending order of destination and the descending order of source). We include two new definitions.

***Definition** 7:* (*prior, superior*) A flow f is *prior* to a flow f' if: (1) $dst_f < dst_{f'}$; (2) $dst_f = dst_{f'}$ and $src_f > src_{f'}$. A flow f is *superior* if no flow is prior to f.

The priority of flows indicates their order to achieve the processing volume of an instance, and superior flows should be processed first because of their small destinations or shorter path lengths. We propose a greedy algorithm in Alg. 2, called Heterogeneous VNF deployment in Line algorithm (HVPL) to solve the deployment problem. We initiate the deployment plan as an empty set in line 1. In lines 2-3, we iteratively select $m(v)$ with the minimum value of $w_m/b_\Omega(m(v))$ to handle superior flows. Then we add the deployment of the

TABLE IV: The values of $cost(m)/b_\Omega(m(v))$.

Ω \ $m(v)$	$m(1)$	$m(2)$	$m(3)$	$m(4)$	$m(5)$	$m'(1)$	$m'(2)$	$m'(3)$	$m'(4)$	$m'(5)$	$m''(1)$	$m''(2)$	$m''(3)$	$m''(4)$	$m''(5)$
$\emptyset$	2	2	2	2	2	3	1.5	1.5	1.5	1.5	4	1	1	2	2
$\{m''(2)\}$	2	∞	2	2	2	3	∞	1.5	1.5	1.5	4	∞	2	2	2
$\{m(3), m''(2)\}$	∞	∞	∞	2	2	∞	∞	∞	3	3	∞	∞	∞	4	4

Algorithm 2 Heterogeneous VNF deployment in Line

In: Sets of vertices V, edges E, flows F and VNFs M;
Out: The deployment plan Ω (initialized to $\emptyset$);

1: **while** not all flows are fully processed **do**
2: Select $m(v)$ with $\min_{\substack{m\in M\\ v\in V}} cost(m(v))/b_\Omega(m(v))$ to handle superior flows;
3: $\Omega = \Omega + m(v)$;
4: **return** The deployment plan Ω.

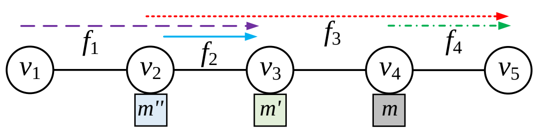


Fig. 2: Illustration of the HVPL algorithm.

new instance to the current plan Ω until all flows are fully served. The deployment plan Ω returns in line 4.

***Theorem** 8:* The worst time complexity of HVPL algorithm is $O(|V|^2 \times |M| \times c_{max})$.

Proof: In each round, we have at most $|V|$ vertices and $|M|$ types of VNFs. The maximum number of rounds is to place VNF instances in every available location in servers, which is $\sum_{v\in V} c_v = O(c_{max} \times |V|)$. Thus, the worst time complexity of HVPL is the maximum number of rounds times the choices in each round, which is $O(|V| \times |M| \times (c_{max} \times |V|) = O(|V|^2 \times |M| \times c_{max})$. ∎

To better understand Alg. 2, we use an example shown in Fig. 2 to illustrate the deployment procedure. In this example, the line topology has 5 vertices with $c_v = 1, \forall v \in V$. We are given a set of heterogeneous VNFs, $M = \{m, m', m''\}$. Their processing capacities are $\alpha = 1, \alpha' = 2$ and $\alpha'' = 4$, and setup costs are $w = 2, w' = 3$, and $w'' = 4$, respectively. There are four flows f_1, f_2, f_3 and f_4, whose paths are shown in Fig. 2 and initial traffic rates are $r_1 = 1, r_2 = 4, r_3 = 1$ and $r_4 = 1$, respectively. The alphabetical order of flows is $f_2 > f_1 > f_4 > f_3$. For each round, we calculate $w_m/b_\Omega(m(v))$, $\forall v \in V, m \in M$. For example, the algorithm is then conducted as: (1) we list all possible deployments over the current empty deployment plan $\Omega = \emptyset$ in the second row of Tab. IV. The smallest one is $w''/b(m''(2)) = 1$. As a result, we deploy a m'' instance on v_2. We prioritize processing the superior flowf_2. (2) referring to the second row of Tab. IV, the smallest one is $w'/b_{\{m''(2)\}}(m'(3)) = 1.5$. As a result, we deploy a m' instance on v_3 to process f_1 and f_4. (3) referring to the third row of Tab. IV, the smallest is $w/b_{\{m''(2), m'(3)\}}(m(4)) = 2$. Thus, we deploy a m instance on v_4 and so all flows are satisfied. We return the feasible deployment plan $\Omega = \{m''(2), m'(3), m(4)\}$.

***Theorem** 9:* The proposed Alg. 2, HVPL, can achieve a deployment with at most $H(\max_{m(v)} b_\emptyset(m(v)))$ times of the minimum cost, where $H(d) = \sum_{i=1}^{d} \frac{1}{i}$.

Proof: Our VNF deployment problem has the same formulation of submodular set cover [19] and the deployment plan Ω is chosen exactly corresponding to its greedy algorithm in Section 2 in [19]. Hence, the approximation ratio follows from Theorem 1 in [19]. $\max_{m(v)} b_\emptyset(m(v))$ is the maximum benefit of only deploying a specific instance m. ($\emptyset$: empty set) ∎

B. Homogeneous VNF deployment in a line topology

***Theorem** 10:* Minimizing the total cost of deployed instances with homogeneous VNFs is also equivalent to minimizing the total amount of wasted processing volume.

Proof: From Theorem 4, $|\Omega|$ is minimized. Because $\sum_{f\in F} r_f$ is a fixed value and α_m is also a constant, $|\Omega| \times \alpha_m - \sum_{f\in F} r_f$, which is the total waste processing volume of deployed VNFs, is also minimized. ∎

Here we further simplify the settings by deploying homogeneous VNF in a line topology. We propose a greedy algorithm, called Greedy VNF Plan (GVP), and prove its optimality for minimizing the deployment cost. The algorithm is shown in Alg. 3. The insight of GVP is to minimize the total processing volume waste based on Theorem 10 when all flows are satisfied. Superior flows are the first to be processed, and GVP only deploys instances when no processing volume is wasted or the superior flow reaches its destination. In GVP, we sort flows in an alphabetical order in line 1. In lines 2-10, we traverse vertices from left to right. When the vertex v has remaining capacities and the total unprocessed traffic rate of superior flows passing v can use up a new instance's processing volume α_m in line 3, we deploy one new instance on v in line 4. In lines 5-9, we handle the case that the superior flows can not use up the processing volume of a new instance. We reallocate the processing volumes of deployed VNFs in line 10 while the deployment plan Ω is returned in line 11.

***Theorem** 11:* The worst time complexity of GVP algorithm is $O(|V| \times c_{max})$.

Proof: We deploy VNFs for $|V|$ vertices, and for each vertex v, we place at most c_v instances. In each loop we place at least one instance in a constant time. The maximum number of loops is $\sum_{v\in V} c_v = O(|V| \times c_{max})$. Thus, the worst time complexity of Alg. 3 is the maximum number of loops, which is $O(|V| \times c_{max})$. ∎

For a better understanding, we use an example shown in Fig. 3 to illustrate the deployment procedure. Each vertex has a capacity of 1, i.e. $c_v = 1, \forall v \in V$. There are four flows f_1, f_2, f_3 and f_4, whose initial traffic rates are $r_1 = 1, r_2 = 4, r_3 = 1$, and $r_4 = 1$, respectively. There is only one type of VNF m with a processing volume $\alpha_m = 2$.

Algorithm 3 Greedy VNF Placement (GVP)

In: VNF m and sets of vertices V, edges E, flows F;
Out: the deployment plan Ω;

```
Sort flows in the alphabetical order;
for each vertex v from 1 to |V| do
    while the sum of unprocessed traffic rate of superior
    flows passing v is no less than α_m and c_v > 0 do
        Allocate one new instance on v;
    if the superior flow f with dst_f ≤ v have some
    unprocessed traffic rate then
        if c_{v'} ≤ 0, ∀src_f ≤ v' ≤ v then
            return Non-existence of a feasible plan;
        else
            Allocate one VNF on max v', ∀c_{v'} > 0, v' < v;
    Reallocate the processing volumes of all deployed
    VNFs from left to right for flows in alphabetical order;
return The deployment plan Ω.
```

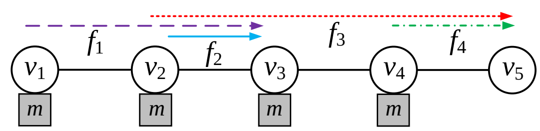


Fig. 3: Illustration of the GVP algorithm.

The alphabetical order of flows is $f_2 > f_1 > f_4 > f_3$. First, we deploy one instance on v_2 because f_2 is the superior flow and its unprocessed traffic rate is larger than α_m. The same happens to v_3 so that we deploy one new instance. f_2 is satisfied and f_1 becomes the superior flow. Since v_2 and v_3 have no remaining capacities, v_1 is the largest vertex with $c_1 > 0$ and one VNF is deployed on v_1. After that, f_4 and f_3 become the superior flows, whose sum of unprocessed traffic rates is larger than α_m on v_4. Then we deploy one VNF on v_4. All flows are satisfied, shown in Fig. 3.

***Lemma** 2:* By applying Alg. 3, a VNF instance on v has some remaining volume only when: suppose f has the lowest priority among all satisfied flows, then no flow f' with $src_{f'} \leq v$ and $dst_{f'} \geq dst_f$ has any unprocessed traffic rate. All flows prior to f' certainly use up the capacity from the next vertex of v to dst_f.

Proof: Alg. 3 deploys a new instance on a vertex v only when: (1) unprocessed traffic rate of superior flows passing v is larger than the processing volume of a VNF instance; (2) one flow f has some unprocessed traffic rate and there is no capacity left from the next vertex of v to dst_f. The first situation has no processing volume waste. In the second situation, the last instance with the remaining processing volume has to be deployed; otherwise, the flow f cannot be satisfied before it reaches its destination, since no vertex capacity is available from v to its destination. ■

***Theorem** 12:* Alg. 3 is optimal for deploying the homogeneous VNF in a line topology.

Proof: We prove the optimality of Alg. 3 by induction. In Theorem 10, we demonstrate that the objective is equivalent to minimizing the total waste of deployed instances. We list all situations that instances have remaining processing volumes. Suppose v_1 is the smallest vertex with such an instance, then all the other instances deployed on and before v_1 have no remaining volume. From Lemma 2, the instance has to be deployed and no more unprocessed traffic rate goes right from the vertex v_1. Thus, there is no deployment that has the waste less than Ω. Assume it is true for all vertices less than v_k, which indicates that no more superior unprocessed traffic rate goes right from the vertex v_k. Then the situation of the next vertex, having an instance with some remaining volume, is the same as the situation of the first vertex v. This is because there is no unprocessed traffic rate of a flow f with $src_f \leq v_k$, making us able to treat the next vertex of v_k as the new origin. Repeatedly, we find the smallest $v > v_k$ with an instance having the remaining capacity. Additionally, we have proven that it is true for the smallest v. So it is also true for the v_{k+1} with an instance having the remaining capacity. To sum up, Alg. 3 has the least amount of wasted volume, which is equivalent to deploying the least number of VNFs. ■

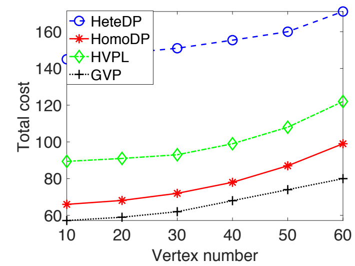


Fig. 4: The impact of topology scale.

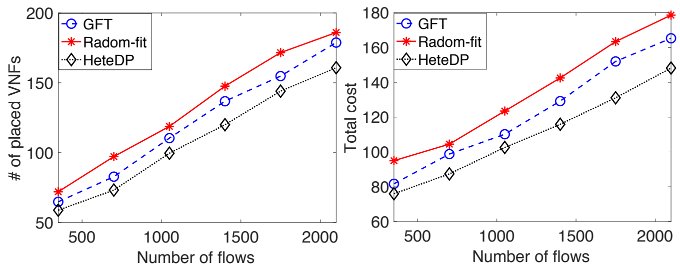
VI. Evaluation

Simulations are conducted to evaluate the performances of our proposed algorithms. After presenting the network and flow settings, the results are shown from different perspectives.

A. Settings

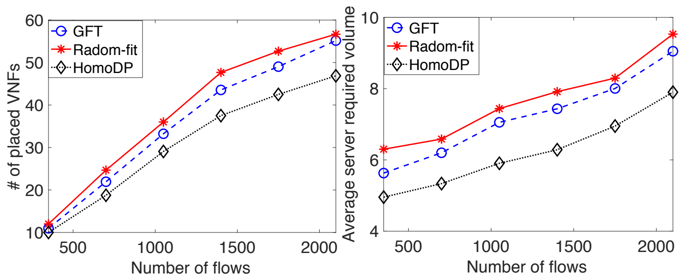
Topology: We test the impact of the topology scale with a fixed flow number of 1000 and basic settings as follows, and the results are shown in Fig. 4. All their total costs have little variance with the vertex number increment. Thus, we only conduct our simulations in a line topology and a random-generated tree topology, both of which empirically have fixed 20 vertices. Each switch vertex is connected to a server with an identical capacity of 10, i.e. $c_v = 10$, $\forall v \in V$. Additionally, traditional data center networks and WAN design over-provision the network with $30-40\%$ average network utilization in order to handle traffic demand changes and failures [20]. As a result, we assume each link has enough bandwidth to hold all flows. This assumption eliminates link congestion and ensures that the transmission of all flows is successful, since routing failure is not the concern in this paper.

VNFs: We conduct the simulations with two sets of VNFs, M and M'. The first set M only includes one type of VNF m, i.e. $M = \{m\}$. Its required server resource is 2, i.e. $w_m = 2$. The processing volume of one m instance is 8, i.e., $\alpha_m = 8$. The second set M' includes three types of VNFs, i.e. $M =$



(a) Total number of deployed VNFs. (b) Total cost.

Fig. 5: Heterogeneous VNF deployment in a tree topology.



(a) Total number of deployed VNFs. (b) Average server required volume.

Fig. 6: Homogeneous VNF deployment in a tree topology.

$\{m, m', m''\}$. Their processing volumes are $\alpha = 6, \alpha' = 8$ and $\alpha'' = 10$, and costs are $w = 1, w' = 2$ and $w'' = 3$.

Traffic: All flows' paths are fixed and their traffic rates are also known a prior. Under the tree topology, the source of each flow is a descendant of its destination. We adopt the flow size distribution of Facebook data centers, which is collected in 10-minute packet traces of three different node types: a Web-server rack, a single cache follower, and a Hadoop node [21]. More than 88% flows are less than 7 Mbps. As a result, the traffic rate ranges from 0.1 to 6 Mbps with a granularity of 0.1 Mbps and is generated randomly in this paper.

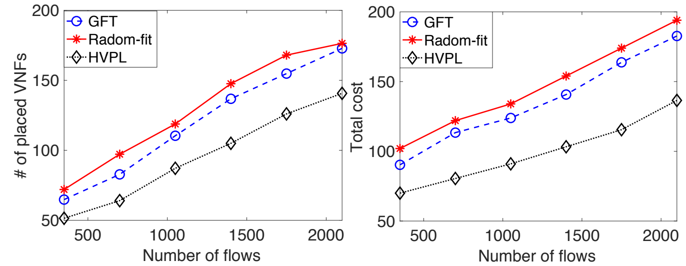
B. Comparison algorithm and performance metrics

We include two benchmark schemes in our simulations:

- Sang et al. [10] propose algorithm GFT for deploying only one type of VNF without the constraint of vertex capacity. VNFs are not deployed until it is the destination of some flows that need to be served.
- *Random-fit* randomly deploys heterogeneous VNFs on random nodes on the paths until all flows are fully served.

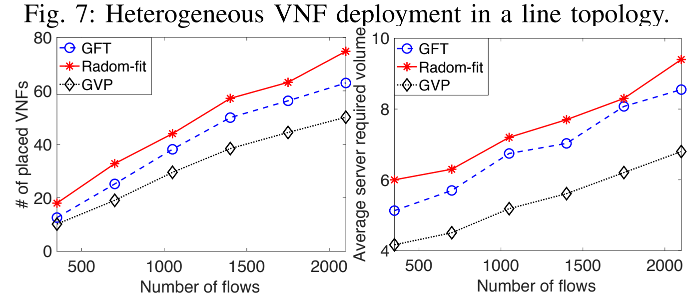
GFT is only designed for deploying the homogeneous VNF instances. When we need to deploy heterogeneous VNFs, we randomly select a single type of VNFs each time and apply GFT to deploy the instances. Additionally, if the vertex capacity is not enough, we simply deploy the VNFs in its nearest descendants with enough remaining capacities until all flows are a hundred percent served.

We use three performance metrics: the total number of deployed instances, the total cost (heterogeneous VNFs), and the average server utilization (homogeneous VNF) for benchmark comparisons. The total number of deployed instances is the sum of deployed VNFs of each type. We also evaluate the total cost corresponding to our objective function as shown in Eq.



(a) Total number of deployed VNFs. (b) Total cost.

Fig. 7: Heterogeneous VNF deployment in a line topology.



(a) Total number of deployed VNFs. (b) Average server required volume.

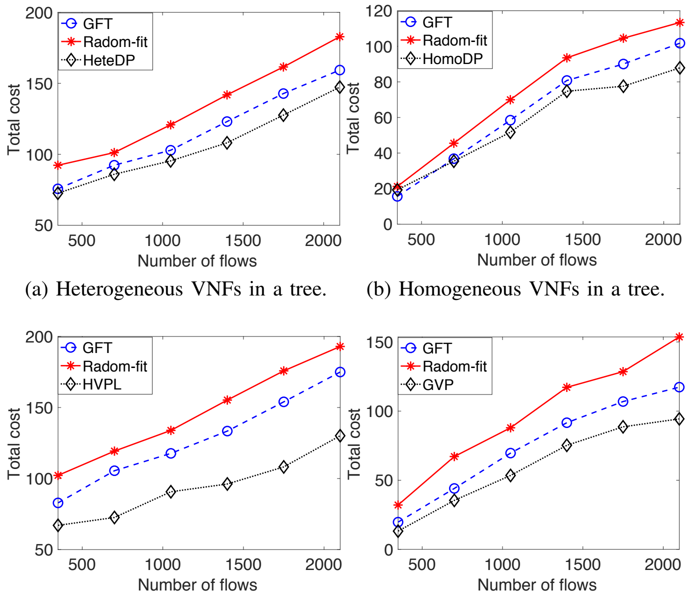
Fig. 8: Homogeneous VNF deployment in a line topology.

(1). Since all vertex capacity settings are identical, the average required server volume is equivalent to the total consumed server volume divided by the total number of servers.

C. Results of the VNF deployment in a tree topology

Fig. 5 shows the results of the heterogeneous VNF deployment in a tree topology. We have tested the algorithms with 350 to 2010 flows. All their sources and destinations are randomly generated. As for the total number of instances, HeteDP deploys the fewest VNFs and outperforms significantly better than the other two as shown in Fig. 5(a). The numbers of deployed instances by the three methods are approximately 3 times the numbers when we only need to deploy a single type of VNFs. In Fig. 5(b), HeteDP has the smallest average server utilization ratio. When there are 2100 flows, HeteDP uses 19.8% less of the total cost than Random-fit and 17.8% less than GFT. This is because HeteDP checks all possible deployment cases and selects the optimal one with the minimum cost. Note that the execution time of HeteDP is tens of GFT and Random-fit because of DP's optimality.

Fig. 6 is the result of homogeneous VNF deployment in a tree topology. We use the same flow set as the one in Fig. 5. The results are shown in Fig. 6(a) and Fig. 6(b), respectively. The number of deployed instances by HomoDP ranges from 11 to 53, which is always much smaller than the other two. When there are 2100 flows, HomoDP deploys 18.5% less VNFs than Random-fit and 16.7% less than GFT. The gap among these three methods becomes larger with more flows involved in the network. We also notice that GFT has a much more similar performance to Random-fit in the general topology. It can be explained that GFT is designed for the tree topology and requires no constraint of vertex capacity. In terms of the average server utilization, HomoDP is at least 17.1% less than the other two no matter how many flows are



(a) Heterogeneous VNFs in a tree. (b) Homogeneous VNFs in a tree.

(c) Heterogeneous VNFs in a line. (d) Homogeneous VNFs in a line.

Fig. 9: Total cost.

generated because HomoDP considers the allocation of the vertex capacity resources.

D. Results of the VNF deployment in a line topolgy

Fig. 7 shows the result of the heterogeneous VNF deployment in a line topology. Alg. HVPL also performs better than GFT and Random-fit. We have tested the algorithms with 350 to 2100 flows. The advantage of our algorithm becomes sharper when there are more flows in the network. This is because it is less possible to waste the spare processing volumes in the deployed VNFs. With more flows, the traffic load is so heavy that the total cost increases significantly. This illustrates that the capacities in all servers are almost used up and more processing volumes of deployed VNFs are wasted. When there are 2100 flows, the total cost of our HVPL algorithm is 32.0% less than Random-fit.

The results of the homogeneous VNF deployment in a line topology are shown in Fig. 8(a) and Fig. 8(b). In Fig. 8(a), the numbers of deployed VNFs by the three methods are approximately one third of the numbers when we only need to deploy heterogeneous VNFs. As the capacity in the server is relatively sufficient, the increasing tendencies of the results are gentle. Our GVP method has the best performance both in the number of deployed VNFs and the average server utilization. The difference is more obvious when the number of flows is larger. This is because GVP is optimal to deploy a single type of VNFs with the constraint of vertex capacity while the other two are not. When there are 2100 flows, GVP deploys 21.6% fewer VNFs than Random-fit and 14.4% fewer than GFT. In terms of the server utilization, GVP always has the least ratio.

E. Results with a larger vertex capacity

To evaluate the impacts of vertex capacity, we enlarge each vertex's capacity from 10 to 20, i.e., $c_v = 20, \forall v \in V$, and other settings remain unchanged. Due to space limitation, we only list the results of the total cost in all four cases of topologies and VNF types of configurations. The basic tendencies of all curves are similar to the results with $c_v = 20, \forall v \in V$. Our algorithms and GFT improve their performances with a smaller total cost. It's worth mentioning that the difference between GFT and each of our algorithms is reduced. This is because a larger vertex capacity is closer to the case without the vertex capacity constraint, where GFT is the optimal solution. However, we find that Random-fit performs even a little worse because of more available locations.

In summary, the simulations verify the correctness and efficiency of our proposed algorithms in the tree and line topologies. They also show that only considering a single type of VNF deployment is too one-sided, because all types of VNFs need to share the limited server resources. It is worth mentioning that our HVPL and GVP can be used as efficient, greedy algorithms with significant insights in all kinds of tree topologies and traffic distributions. Additionally, the general topologies can also be transformed to the combination of several trees by grouping flows and then apply our algorithms.

VII. Conclusion

We study the joint VNF deployment and flow allocation problem. We aim at minimizing the total cost of deploying VNF instances when all flows are fully processed. We assume that all flows request the same type of network functions. We study the heterogeneous VNF deployment in tree topologies. First, we prove the NP-hardness of the deployment and propose a DP solution. Then we introduce an improved DP solution for homogeneous VNFs in a tree topology. We reformulate the deployment of heterogeneous VNFs in a line and propose a performance-guaranteed strategy. An optimal greedy solution is designed for homogeneous VNF deployment in a line.

It is worth mentioning that the vertex capacity constraint in terms of the maximum number of VNF instances can be extended to a constraint on the total resource capacity. Setting up each type of VNF instance needs different amounts of the vertex resource besides different setup costs. Hence, our DP solution, HeteDP, needs to include one more dimension of the available resource in the current vertex. In this case, the *Deploy* item in the DP formulation becomes a 2-D knapsack problem. Although the extension can still be addressed in a DP formulation, we leave detailed treatment to our future work.

VIII. Acknowledgment

This research was supported in part by NSF grants CNS 1629746, CNS-1651947, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and CNS 1439672.

Appendix
Proof of Theorem 1

Here we prove our theorem 1. First, checking the feasibility of a deployment plan is in a polynomial time, since we can check in $O(|F|)$ time to make sure that all flows are fully processed before their destinations.

Second, we show that *Unbounded Subset Sum* [22] is reducible to the *heterogeneous VNF deployment*. Consider a case of *Unbounded Subset Sum* with n numbers $W = \{w_1, w_2, ..., w_n\}$ and a target w. In constructing an equivalent case of the *heterogeneous VNF deployment*, we simplify the deployment problem by having a line topology with unlimited-capacity vertices. We are given a set of flows F, all of whose source and destination are the leftmost and rightmost nodes in the line. Each flow has an initial traffic rate r_f and requests the same network function. We assume the total traffic rate $\sum_{f \in F} r_f$ is equal the target w of the *Unbounded Subset Sum*, i.e. $\sum_{f \in F} r_f = w$. We are given a set of VNF types M with n types for the requested network function. The setup costs of the VNF types are $w_1, w_2, ..., w_n$, and their processing volumes are the same to the setup costs, meaning $\alpha_i = w_i$. The sum of the processing volumes of the deployed VNF instances should be no less than w since all flows needs to be fully processed. When there is no processing volume wasted in a deployment plan, the sum of the processing volumes is exactly w. The total cost of the deployment is $\sum \alpha_j = \sum w_j = w$, which is also the minimum. If we can find such a deployment of VNFs with the costs of $w'_1, w'_2, ..., w'_k$ adding up to the total cost w, then the corresponding numbers in the *Unbounded Subset Sum* instance can also add up to exactly w.

Conversely, if there are numbers $w'_1, w'_2, ..., w'_k \in W$ adding up to exactly w in the *Unbounded Subset Sum*, then we can deploy the corresponding VNF instances with setup costs $w'_1, w'_2, ..., w'_k$; this is a feasible deployment plan with the minimal total cost w. Consequently, since the *Unbounded Subset Sum* is an NP-complete problem, our *heterogeneous VNF deployment* is NP-hard. The theorem holds.

REFERENCES

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *SIGCOMM 2012*.

[2] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," in *SIGCOMM 2017*.

[3] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *SIGCOMM 2014*.

[4] J. Sherry, S. Ratnasamy, and J. S. At, "A survey of enterprise middlebox deployments," in *Semantic Scholar*, 2012.

[5] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *NSDI 2014*.

[6] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *SIGCOMM 2017*.

[7] S. Seyyedi and B. Akbari, "Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes," in *CNDS 2011*.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[9] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: A high performance, server-centric network architecture for modular data centers," in *SIGCOMM 2009*.

[10] 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 *INFOCOM 2017*.

[11] P. Duan, Q. Li, Y. Jiang, and S. T. Xia, "Toward latency-aware dynamic middlebox scheduling," in *ICCCN 2015*.

[12] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *PRESTO 2010*.

[13] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *INFOCOM 2017*.

[14] M. Rost and S. Schmid, "Np-completeness and inapproximability of the virtual network embedding problem and its variants," Technical Report, Tech. Rep.

[15] ——, "Virtual network embedding approximations: Leveraging randomized rounding," *arXiv preprint arXiv:1803.03622*, 2018.

[16] S. Martello, "Knapsack problems: algorithms and computer implementations," *Wiley-Interscience series in discrete mathematics and optimization*, 1990.

[17] G. B. Mathews, "On the partition of numbers," *Proceedings of the London Mathematical Society*, vol. s1-28, no. 1, pp. 486–490, 1896.

[18] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[19] L. A. Wolsey, "An analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, vol. 2, no. 4, pp. 385–393, Dec 1982.

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *SIGCOMM 2013*.

[21] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM 2015*.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.