# On the Theory of Function Placement and Chaining for Network Function Virtualization

Jinbei Zhang
Sun Yat-sen University
jinbeizhang1988@gmail.com

Weijie Wu
2012 Labs, Huawei
wuweijie2@huawei.com

John C.S. Lui
Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

## ABSTRACT

Network function virtualization (NFV) can significantly reduce the operation cost and speed up the deployment for network services to markets. Under NFV, a network service is composed by a chain of ordered virtual functions, or we call a "network function chain." A fundamental question is when given a number of network function chains, on *which servers should we place these functions* and how should we *form a chain* on these functions? This is challenging due to the intricate dependency relationship of functions and the intrinsic complex nature of the optimization. In this paper, we formulate the function placement and chaining problem as an integer optimization, where each variable is an indicator whether one service chain can be deployed on a configuration (or a possible function placement of a service chain). While this problem is generally NP-hard, our contribution is to show that it can be mapped to an exponential number of min-cost flow problems. Instead of solving all the min-cost problems, one can select a small number of mapped min-cost problems, which are likely to have a low cost. To achieve this, we relax the integer problem into a fractional linear problem, and theoretically prove that the fractional solutions possess some desirable properties, i.e., the number and the utilization of selected configurations can be upper and lower bounded, respectively. Based on such properties, we determine some "good" configurations selected from the fractional solution and determine the mapped min-cost flow problem, and this helps us to develop efficient algorithms for network function placement and chaining. Via extensive simulations, we show that our algorithms significantly outperform state-of-art algorithms and achieve near-optimal performance.

## 1 INTRODUCTION

Network function virtualization (NFV) is a promising trend to carry out network services. Different from traditional approaches, NFV uses general-purpose servers and functions are deployed as software components. Virtual functions can be quickly instantiated or deallocated from servers, thereby dramatically reducing the deployment time to the market [1]. NFV is based on virtual network functions (vNFs). Each virtual network function is a software component which performs a specific task (or function), e.g., a firewall or a packet inspection unit [2–4]. To provide a network service, several vNFs need to be chained and executed in order, which is called a "service chain". For example, some data flows may need to first go through a firewall function, then a packet inspection function, and finally a packet processing function. Based on a number of general functions deployed on servers, an incoming service request can be executed by choosing the necessary functions and chaining them together in a flexible and on-demand manner. Some key design questions are, which servers should we place these functions (i.e., function placement), and how we can "group" them to process network request (i.e., function chain formation) in a low-cost and high-efficiency manner?

The above two design questions are non-trivial due to the complex nature of the design space and objectives. In general, a set of connected servers are responsible to host the virtual network functions. Each server has a capacity constraint, i.e., it can only host a limited number of functions. Each pair of servers $(i, j)$ may (or may not) be connected, i.e., the output of a function deployed in server $i$ can (or cannot) be used as the input to a function in server $j$. In other words, a service chain formation has to satisfy the connectivity constraint: It needs to choose functions deployed on a sequence of connected pair of servers. The connectivity depends on the physical interconnection of servers, and the dynamic availability of link bandwidth. Furthermore, given a pair of connected servers, there is a link cost between them, which may represent the bandwidth cost or transmission delay. The objectives of a NFV placement and chaining problem may include: (1) to serve as many incoming requests as possible; (2) to reduce the operating cost (e.g., bandwidth cost), and (3) to improve the service quality (e.g., reducing task processing time), subject to the capacity and connectivity constraints. Such a problem is complicated because (1) there are multiple (and possibly conflicting) objectives, (2) a decision on the function placement and chaining is an integer programming problem, which has a high computational complexity in general, and (3) the constraints are complicated and dynamic, subject to the stochastic nature of request arrivals and departures.

There have been considerable efforts to address the aforementioned design issues. State-of-the-art approaches can be classified into two categories. The first one is using heuristic schemes [5–8]. For example, one heuristic approach is to consider NFV placement requests in an arbitrary sequence, and greedily form a service chain with the lowest possible cost in each assignment. These heuristics are often simple, but the performance is not guaranteed and more importantly, the underlying reason for such heuristics is unclear. The second approach is to formulate an integer optimization for the

*ACM Mobihoc '18, June 26-29, 2018, Los Angeles, CA, USA*
2018. ISBN 978-1-4503-5770-8/18/06. . . $15.00
DOI: https://doi.org/10.1145/3209582.3209592

function placement and chaining problem, relax it to a linear programming, and perform random rounding on the fractional solution to obtain an integer solution [9, 10], so that the function placement and chaining can be done according to this integer solution. Such approaches are often believed to be closer to the optimality of a pre-defined objective, yet, a random rounding can easily violate some constraints (e.g., the capacity of a server, please refer to section 5.1). The violation can be severe when the size of the problem scales up, and thus, they have a fundamental scalability issue.

In this paper, we formulate an integer optimization for the function placement and chaining problem. We first show its high complexity and map it to min-cost flow problems. To reduce the complexity, we relax it into a linear programming problem, from which we obtain fractional solutions. Different from existing approaches which process the fractional solution in a straightforward rounding manner, we analyze the features of such fractional solutions, i.e., the number and the utilization of selected configurations (function placements for all service chains) can be upper bounded and lower bounded, respectively. One important property of these bounds is that they are only dependent on the parameters of service requests, but independent of the network size. Based on these bounds, we design *time-efficient*, *near-optimal* and *scalable* algorithms by taking the advantage of some "good" fractional configurations that are selected by the criterion we propose. To summarize, our contributions are:

- We set up a formal mathematical model to capture the function placement and chaining problem for network function virtualization, and show it can be mapped to min-cost flow problems.
- We relax the integer optimization into a linear programm and reveal some attractive properties of its fractional solution, i.e., the number and the utilization of selected configurations can be upper bounded and lower bounded, respectively.
- Based on such properties, we propose efficient algorithms by selecting a small number of min-cost flow problems to determine network function placement and chaining, which are shown to achieve superior performance over existing approaches via extensive simulations.

This is the organization of the paper. In Section II, we provide a model for NFV function placement and chaining. In Section III, we map the problem to an exponential number of min-cost problems. In Section IV, we relax the integer optimization into a linear programming problem, analyze the properties of its fractional solutions, and propose an efficient algorithm for function placement and chaining. In Section V, we evaluate the performance of our proposed algorithms via extensive simulations. We conclude our work in Section VI.

## 2 NETWORK MODEL

In this section, we present the network model of placing a number of service chains on a physical network, and our objective is to minimize the total cost, including link cost and server cost.

**Server model.** We consider a connected graph $G = (V, E)$, where $V$ is the set of servers, and $E$ is the set of links. For each server $i \in V$, its capacity is an integer $c_i$, which indicates that server $i$ can host at most $c_i$ functions. The cost to implement a function on server $i$ is denoted as $w_i$. The number of servers is $|V| = N$. For each link $e \in E$, its capacity is an integer $c_e$. When there are two functions communicating through link $e$, they will consume one unit of link capacity and there is a cost $w_e$ for using the unit link capacity. Physically, the link cost can represent bandwidth cost or data transmission delay between these two servers.

**Service requests.** When service requests arrive, the network will determine the types of service chains to serve the service. A service chain may need to be implemented with multiple instances depending on the traffic requesting the service. In general, we assume that a total of $K$ service chains needed to be deployed. Under the NFV paradigm, each service chain is composed of a number of ordered virtual network functions. Let us assume we need $L_k$ functions for the $k$-th service chain, and that $L_1 \le L_2 \le ... \le L_K = L$. Note that a function can only be used by one service. If two services need the same function, we need two instances of this function. The number of functions in a service chain is usually not very large, and we assume $L_K$ is a constant independent of the number of service chains or servers.

**Service chain forming.** For service chain $k$, there are $L_k$ functions. Since each function can be placed in any of the $N$ servers, there are at most $N^{L_k}$ configurations for service chain $k$, where each configuration is one possible placement and chaining scheme for a service chain. For configuration $C_{k,p_k}$ $(1 \le p_k \le N^{L_k})$, functions are connected by $L_k$ servers and $L_k - 1$ links[1], and the cost of this configuration $W_{k,p_k}$ is the total cost to implement these $L_k$ functions and use $L_k - 1$ links, i.e., $W_{k,p_k} = \sum_{i \in C_{k,p_k}} w_i + \sum_{e \in C_{k,p_k}} w_e$. Note that when the capacity of one server is larger than two, it is possible that two or more functions of a service chain can be placed on this server.

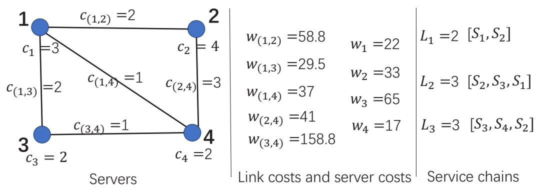


**Figure 1: An example of network model.** $c_i$ ($c_{(i,j)}$) **is the capacity of server** $i$ **(link** $(i,j)$**),** $w_i$ ($w_{(i,j)}$) **is the cost for unit capacity at server** $i$ **(link** $(i,j)$**), and assume** $w_{(i,i)} = 0$**. There are three service chains to be deployed, each with** $L_k$ $(k = 1, 2, 3)$ **functions.**

[1]Each link here is a physical link, which assumes that nearby functions on a same chain should be placed on two physically connected nodes. We note that this assumption can be readily extended to more general scenarios, e.g., a link represents a shortest path between two nodes.

One example of the network model is presented in Figure 1, where each $S_i$ is a function in a service chain. One possible configuration for the third service chain is $C_{3,p_3} = \{3, 1, 4\}$, where functions $S_3$, $S_4$ and $S_2$ are placed on servers 3, 1, 4, respectively. The cost for this configuration is $W_{3,p_3} = w_3 + w_1 + w_4 + w_{(3,1)} + w_{(1,4)} = 170.5$. The question is can we deploy all the three service chains in these servers with the minimum cost? Now let us formulate the problem and present the objective function formally, i.e., to place functions of all services in the physical network, and form chains for them with the minimum total cost[2]. Define $z_{k,p_k} = 1$ if service $k$ chooses configuration $C_{k,p_k}$, and $z_{k,p_k} = 0$ otherwise. To reduce its total cost, we formulate the following integer optimization problem:

$$\text{Problem } \mathbf{A}: \quad \min_{z} \sum_{k=1}^{K} \sum_{p_k=1}^{N^{L_k}} W_{k,p_k} \cdot z_{k,p_k}, \tag{1}$$

with constraints,

$$\sum_{p_k=1}^{N^{L_k}} z_{k,p_k} = 1 \ (1 \le k \le K), \tag{2}$$

$$\sum_{k=1}^{K} \sum_{p_k=1}^{N^{L_k}} f(i, p_k) \cdot z_{k,p_k} \le c_i \ (1 \le i \le N), \tag{3}$$

$$\sum_{k=1}^{K} \sum_{p_k=1}^{N^{L_k}} f(e, p_k) \cdot z_{k,p_k} \le c_e \ (e \in E), \tag{4}$$

$$z_{k,p_k} \in \{0, 1\}, \tag{5}$$

where $f(i, p_k)$ is the number of functions placed in server $i$ for configuration $C_{k,p_k}$ and $f(e, p_k)$ is the number of times that configuration $C_{k,p_k}$ will traverse through link $e$. Constraint (2) ensures that every service chain is satisfied; (3) ensures the feasibility of the server capacity, (4) ensures the feasibility of the link capacity and (5) is the integer constraint. Note that the capacity information in (3) and (4) represents the *current* network status. Thus, when the servers are already running some network services, by quickly solving the optimization, we can decide the placement and chaining for these newly arrived network requests without affecting the ones in execution.

## 3 ANALYSIS ON THE INTEGER PROBLEM

In this section, we conduct analysis on the integer problem **A**. In general, the problem is NP-hard, as shown in existing works [8–10]. Here, we want to further point out that, when link constraint (4) is considered, finding a feasible solution is already NP-hard.

[2]In this formulation, we decide the function placement and chaining simultaneously. This is consistent with existing works [4–8]. In reality, physical function installation in servers may be slow; instead, one can pre-install all possible functions in a server in advance, but dynamically activate them upon service request arrivals, subject to the server's computational capability. Physically, a function placement corresponds to a function activation in this case.

**Lemma** 1. *When link constraint* (4) *is considered, it is NP-hard to find the maximal number of service chains that can be deployed for problem* **A**.

PROOF. We will degenerate problem **A** into a maximal set packing problem. To see this, assume each server has only unit capacity, and each link is either of unit capacity or disconnected. Then a configuration can be seen as a set of several nodes, and two configurations that share a same node can not be chosen together.

Therefore, this degenerated problem of **A** can be seen in the class of maximal set packing problems [16], which is surely NP-complete. □

Lemma 1 shows that it is difficult to know how many service chains can be deployed. Thus, it could be meaningless to consider the cost minimization problem **A** when the number of deployed service chains is unknown. To ensure the feasibility of problem **A**, in this section we neglect the link capacity constraint (4), and assume the servers are fully connected. This simplification maybe valid in data center networks where the link bandwidth is large and all servers are "reachable" by going through multiple physical links. Thus, the chain deployment is only subject to the server capacity constraint. It is easy to verify that when the capacity summation of all servers is larger than the total number of functions to be deployed, i.e., $\sum_{i=1}^{N} c_i \ge \sum_{k=1}^{K} L_k$, problem **A** with respect to constraints (2), (3), (5) has feasible integer solutions. We denote this problem as problem $\mathbf{A}'$.

***Remark:*** We note that the assumption on link capacity constraints in this section is mainly due to the guarantee for a feasible solution of the integer problem. This assumption will not affect the properties of the relaxed linear problem that we will investigate in Section IV, and will not affect the scheme we proposed based on the desired properties.

### 3.1 Problem re-formulation

Before we map the integer problem to min-cost flow problems (MCFP), let us first re-formulate problem $\mathbf{A}'$ to an equivalent form, such that each server can host exactly one function. By doing so, multiple functions deployed on a same server can be decoupled.

Recall that the original physical network is $G = (V, E)$, where server $i$ has a capacity of $c_i$. Let us map each server $i \in V$ into a *subset* $V_i$, consisting of $c_i$ servers, each with a unit capacity. For each link $e = (i, j)$ $(j \ne i)$ with link cost $w_e$ in $E$, the constructed model will have links between every server in $V_i$ and every server in $V_j$, with the same link cost $w_e$. For each self-loop link $e = (i, i)$ with cost $w_e$ in $E$, the constructed model will have links among every two server in $V_i$ with the same link cost $w_e$. By such construction, the new network model is denoted as $G' = (V', E')$, where the total number of servers is $|V'| = \sum_{i=1}^{N} c_i$. By so doing, the optimal solution to the original network with arbitrary capacities can be mapped to the optimal solution to the newly constructed network $G'$ with nodes of unit capacity, and their minimum costs are the same. Therefore, we have the following lemma.

**Lemma** 2. *Let $cost_I(G)$ and $cost_I(G')$ denote the minimal cost for the integer solution to place the $K$ service chains in $G$ and $G'$, respectively. We have*

$$cost_I(G) = cost_I(G'). \tag{6}$$

## 3.2 Mapping to MCFP

While ignoring the link capacity constraint and transforming the original network into servers with unit capacity $G'$, the cost minimization problem can be seen as finding $K$ paths with minimum cost in a network, which is still NP-hard [11].

**Lemma** 3. *Problem $\mathbf{A}'$ is NP-hard when $L \geq 3$.*

We now map problem $\mathbf{A}'$ to min-cost flow problems. Recall that $|V'|$ is the total capacity of $N$ servers, i.e., $|V'| = \sum_{i=1}^{N} c_i$.

**Theorem** 1. *Problem $\mathbf{A}'$ can be mapped to at most $L^{|V'|}$ min-cost flow problems.*

Proof. When an optimal solution $z$ for $G'$ exists, i.e., $\sum_{k=1}^{K} L_k \leq \sum_{i=1}^{N} c_i$, assume the solution places service chain $k$ ($k = 1, 2, ..., K$) in configuration $C_{k,p_k} = \{v_{k1}, v_{k2}, ..., v_{kL_k}\}$, where $v_{kj}$ is a server in $V'$ in which the $j$-th function of service chain $k$ is placed. Now based on $G'$, we construct a directed graph $G_d$ as follows.

First, let $T_i = \{v_{ki} | k = 1, 2, ..., K\}$, which contains the $i$-th server ($i = 1, 2, ..., L_k$) in each $C_{k,p_k}$ of $z$. For each node in $V'$ but not in any $T_i$, randomly assign it to one $T_i$. Second, add a node $s$ with zero node cost, and connect $s$ to each node in $T_1$ with a zero cost link. Node $s$ can then be seen as a virtual source. Third, for each service chain $k$, add a node $d_k$ with zero node cost, which can be seen as a virtual destination for service chain $k$. Connect each node in $T_{L_k}$ to $d_k$ with a link of zero cost. Lastly, for $i = 1, ..., L_K - 1$, connect each node in $T_i$ to each node in $T_{i+1}$ with a link, whose cost is the same as the corresponding link in $E$.

An example of the construction is shown in Figure 2. There are 13 servers of unit capacity and three service chains needed to be deployed. Service chain 1 has two functions, while service chains 2 and 3 both have 3 functions. Let's assume the optimal integer solution exists. In the solution, the first and second function of service chain 1 are placed in server 1 and server 5, respectively. The functions of service chain 2 are placed in servers $6, 7, 8$ in order, while the functions of service chain 3 are placed in servers $3, 2, 9$ in order. Based on this optimal solution, we construct a directed graph as described. We put the servers hosting the first function of each service chain into $T_1$ (servers $1, 6, 3$), put the servers hosting the second function of each service chain into $T_2$ (servers $5, 7, 2$), and put the servers hosting the third function of each service chain into $T_2$ (servers $8, 9$). Other servers are randomly assigned to one of the $T_i$. In the constructed graph, there is no link between any two nodes in the same $T_i$, and links only exist between nearby $T_i$. In this sense, we say that $T_i$ is in layer $i$.

Now if the source node $s$ wants to send a unit flow to each $d_k$ through the directed network $G_d$, the problem turns into

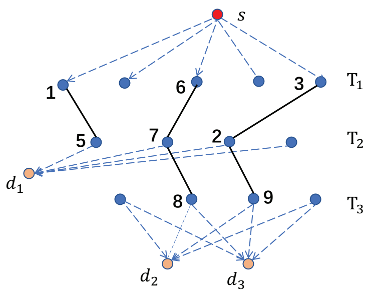


**Figure 2: An example of constructing min-cost flow graph.**

a well-known min-cost flow problem. By solving the min-cost flow problem[3], we can obtain a solution $z'$. Due to the nature of the min-cost flow problem and the constructed graph $G_d$, $z'$ must be a feasible solution to the integer problem $\mathbf{A}'$, where the servers to host the $i$-th functions of service chains are chosen from $T_i$. Furthermore, while it cannot be guaranteed that $z' = z$, the cost of $z'$ must be the same as that of $z$, i.e., $z'$ is also an optimal solution to problem $\mathbf{A}'$. Otherwise, $z$ will not be an optimal solution to $\mathbf{A}'$.

On the other hand, as long as there is an optimal solution $z$ to $\mathbf{A}'$, we can construct the corresponding min-cost flow problem. Therefore, if we are able to list all the possible $G_d$, the achievable deployment with the minimum cost can be found. Note that each server of unit capacity can be at any layer and $L_K = L$. The total number of $G_d$ can then be upper bounded as $L^{|V'|}$.

Therefore, we conclude this theorem. $\square$

***Insight:*** Theorem 1 establishes the relationship between our problem $\mathbf{A}'$ and min-cost flow problems. In fact, when the link capacity constraint is considered, problem $\mathbf{A}$ can still be mapped to min-cost flow problems. The difference lies in Lemma 2 when the server with multiple capacity is to be divided into multiple servers with unit capacity. When link capacity is considered, the link capacity also needs to be divided into multiple unit ones and assigned to different server pairs. Then, if a scheme can construct a small number of min-cost flow problems which are likely to have low costs, the scheme can then achieve good performance with low complexity. Section IV is devoted to such analysis and schemes.

## 4 AN EFFICIENT ASSIGNMENT SCHEME

In the previous section, we show that the integer problem $\mathbf{A}$ with link capacity constraints or even its simplified version $\mathbf{A}'$ without link capacity constraints are both NP-hard and

[3]The min-cost flow problem originally only considers link costs. However, it can accommodate to consider server costs with a slightly modification in our case, i.e., for each link $(i, j)$, we can add the link cost with $0.5(c_i + c_j)$. Then, the server costs are mapped to the link costs.

difficult to solve. We now propose an efficient scheme for problem **A**, which can achieve near-optimal performance.

The overall idea is as follows. We first relax the integer problem **A** into a linear problem **B** (specified later) which can be solved efficiently. From the solution of the linear problem **B**, we will choose some configurations which have low costs. We then map these configurations into layers as we have shown in the proof of Theorem 1 and run the min-cost flow optimization to obtain the final assignment solution for problem **A**.

Note that this approach is *different* from the conventional greedy algorithms which choose the configurations with the lowest cost one by one, and also *different* from the random rounding approach which may violate the server's capacity. Our approach starts from showing some performance guarantees for the relaxed linear problem, based on which we are able to determine configurations with near-optimal performances.

## 4.1 Integer problem relaxation

We first relax the original integer problem (1) into a linear fractional problem (LP) by relaxing the constraint (5) to

$$z_{k,p_k} \in [0,1]. \tag{7}$$

When relaxing the integer constraint, we need to make sure the capacity constraints are not violated, i.e., for each configuration, the number of functions deployed on a server cannot exceed the server's capacity, and the link capacity constraints are satisfied. To achieve this, we add the following two constraints,

$$z_{k,p_k} = 0, \text{ if } f(i,p_k) > c_i. \tag{8}$$

$$z_{k,p_k} = 0, \text{ if } f(e,p_k) > c_e. \tag{9}$$

We then investigate the linear problem (1) with constraints (2), (3), (7), (8) and (9), which we called problem **B**. When $z_{k,p_k} > 0$ in a solution, this value is called the "utilization" of configuration $C_{k,p_k}$, which means service chain $k$ is placed on configuration $C_{k,p_k}$ with a fraction of $z_{k,p_k}$. Note that while $z_{k,p_k}$ is derived from problem **B**, each configuration $C_{k,p_k}$ with $z_{k,p_k} > 0$ is a feasible deployment for problem **A**, due to the introduction of (8) and (9).

While we are interested in the optimal fractional solutions, there could exist many solutions with the same minimum cost. For simplicity of analysis, we will only focus on the extreme points of feasible solutions. A feasible solution is an extreme point if it cannot be written as a convex combination of other feasible solutions [13]. It can be easily verified that if the linear problem has optimal solutions, there must exist extreme points which are optimal. Therefore, without loss of generality, when we say a solution is optimal in this paper, the solution is also an extreme point, unless stated otherwise.

Now we study the property of the optimal fractional solution for problem **B**. We first assume that all services have the same number of functions, i.e., $L$. This assumption will be relaxed later. Our objective then turns into selecting $K$ configurations with the minimum cost (problem **B′**). We have

$$\text{Obj:} \qquad \min_{z} \sum_{k=1}^{K} \sum_{p_k=1}^{N^L} W_{k,p_k} \cdot z_{k,p_k} \tag{10}$$

$$\begin{aligned} \text{s.t.} \qquad & \sum_{p_k=1}^{N^L} z_{k,p_k} = 1, \ (1 \le k \le K) \\ & \sum_{p=1}^{N^L} f(i,p_k) \cdot z_{k,p_k} \le c_i \ (1 \le i \le N), \\ & \sum_{p=1}^{N^L} f(e,p_k) \cdot z_{k,p_k} \le c_e \ (e \in E), \\ & 0 \le z_{k,p_k} \le 1. \end{aligned} \tag{11}$$

Note that some configurations may violate the capacity constraints, and the utilization on these configurations will be set to 0 directly, i.e., $z_{k,p_k} = 0, \quad \text{if} \quad f(i,p_k) > c_i$ or $f(e,p_k) > c_e$.

Let $z = [z_{1,1}, ..., z_{1,N^L}, z_{2,1}, ..., z_{2,N^L}, ..., z_{K,1}, ..., z_{K,N^L}]^T$, and $W$ be the corresponding cost vector for $z$. We can rewrite (10) and (11) into a matrix form:

$$\text{Obj:} \qquad \min_{z} W \cdot z \tag{12}$$

$$\text{s.t.} \qquad H \cdot z \le b, \tag{13}$$

where $H$ is a $(K + N + |E| + 2N^L) \times KN^L$ matrix, and $b$ is a $(K + N + |E| + 2N^L) \times 1$ matrix. Denote $a_j$ and $b_j$ as the $j$-th row of $H$ and $b$, respectively. For a feasible solution $z^*$, if $a_j z^* = b_j$ (or $a_j z^* < b_j$), we say that the $j$-th inequality is active (or inactive) at $z^*$.

Intuitively, the number of non-zero elements at an extreme point will be upper bounded by the number of active inequalities. While the number of inequalities in problem **B′** is $\Theta(N^L)$, we will show below that the number of non-zero elements is upper bounded by $O(LK)$.

Denote the minimum link capacity as $c_l = \min_{e \in E} c_e$ ($c_e \ge 1$). Denote the minimum server capacity as $c_s = \min_{i=1,2,...,N} c_i$ ($c_i \ge 1$). As the number of service chains is $K$ and each service chain has $L$ functions, the number of servers that are fully used is at most $\frac{LK}{c_s}$. As each service chain will at most go through $(L-1)$ links, the number of links that are fully used is at most $\frac{K(L-1)}{c_l}$. If we can upper bound the number of active equalities in the constraints ($z_{k,p_k} \in [0,1]$), the number of utilized configurations in one extreme point can also be upper bounded. We then have the following result.

**Lemma** 4. *If $z$ is an optimal solution to problem **B′**, the number of non-zero elements in $z$ is upper bounded by* $K + \frac{LK}{c_s} + \frac{K(L-1)}{c_l}$.

The proof of Lemma 4 is presented in the appendix. Based on Lemma 4, we will later make use of these utilized configurations in our proposed scheme, instead of searching for the exponential (e.g., $L^{|V'|}$) assignments of MCFP as we discussed in Section III. Furthermore, the fraction of service chains deployed on some selected configurations cannot be all trivial, e.g., the fraction will not approach zero even when the network scales. Denote $c_m = \min(c_s, c_l)$.

**Theorem** 2. *For the configurations that constitute an optimal solution $z$ of problem $\mathbf{B}'$, there are at least $\alpha \cdot K$ of them, with utilization no smaller than $\frac{1-\alpha}{1+2L/c_m-\alpha}$, where $\alpha$ can be an arbitrary value in $[0,1]$.*

Proof. Assume the number of configurations $n_p$ whose utilization is larger than $\frac{1-\alpha}{1+2L/c_m-\alpha}$ is smaller than $\alpha K$. According to Lemma 4, the number of configurations whose utilization is no larger than $\frac{1-\alpha}{1+2L/c_m-\alpha}$ is at most $K+\frac{2KL-K}{c_m}-n_p$. The total number of service chains that are deployed can be bounded by $n_p \cdot 1+(K+\frac{2KL-K}{c_m}-n_p)\cdot\frac{1-\alpha}{1+2L/c_m-\alpha}$, which can be verified to be smaller than $K$. This is contradictory with the fact that the total number of services is $K$. Therefore, we reach the conclusion. □

***Remark:*** Lemma 4 and Theorem 2 have the following important implications. i) The number of used configurations will decrease as the minimum capacity $c_m$ increases, while the utilizations will increase as $c_m$ increases. ii) If capacity violation is allowed similar to the random rounding [9, 10], our result suggests a constant violation gap. To see this, note that the number of functions ($L$) in a service chain is a constant and usually small, e.g., less than 10. Hence, if we choose $\alpha = 0.5$, there are at least $K/2$ configurations with a constant utilization, i.e., $v = \frac{1-\alpha}{1+2L/c_m-\alpha}$. We then choose these $K/2$ configurations which have the highest utilizations to deploy the service chains. Then the capacity violation gap will be upper bounded by $\frac{1}{v}$, so we can place the $K$ service chains with a cost of $\frac{1}{v}$ times the minimum possible cost, if the capacity of each node and every link is expanded to $\frac{1}{v}$ times. For comparison, random rounding may introduce an unbounded violation gap as the network scales. iii) From a practical point of view, our simulation result suggests a much better property than that demonstrated in the theorem, e.g., more configurations are assigned with higher utilizations. Therefore, we can make use of these good configurations (e.g., those with high utilizations) to deploy service chains. Further detail is presented in the next subsection.

Note that Lemma 4 and Theorem 2 assume the same number of functions. The following result considers service chains that have different number of functions for problem $\mathbf{B}$.

**Corollary** 1. *For the configurations that constitute an optimal solution $z$ in problem $\mathbf{B}$, there are at least $\alpha \cdot K$ of them, with utilization no smaller than $\frac{1-\alpha}{1+2L/c_m-\alpha}$, where $L = \left(\sum_{k=1}^{K} L_k\right)/K$ and $\alpha$ can be an arbitrary value in $[0,1]$.*

Note that $L$ can be seen as the average number of functions for the $K$ service chains, which will be used to bound the number of inequalities for capacity constraints. Since the proof is similar to that of Theorem 2, it is omitted here.

According to Corollary 1, if the server capacity is allowed to be violated, the violation gap is also upper bounded by a constant.

## 4.2 Mapping fractional solution into integer solution

In the previous subsection, we show the desired properties of the linear problem for function chain placement. In this subsection, we make use of the good configurations indicated by the fractional solution to obtain the integer solution for the general chaining placement problem $\mathbf{A}$.

The good configurations are selected and mapped to layers in steps 2-12, as shown in Algorithm I. We first solve a linear problem (step 3), and then choose the maximal independent configuration set in steps 5-11 (each configuration corresponds to a function placement for a service chain). With higher priority, we choose the configuration for service chains with a larger number of function (step 5). The selected configuration is mapped to layers in step 8. Then, the scheme will update the link capacity and server capacity (step 9), i.e., reduce the server capacity and link capacity used by previous selected configurations, and choose configurations for other service chains (back to step 5). When the maximal independent configuration set is found and there are unselected service chains, the scheme (back to step 2) will then try to place the remaining service chains with the remaining server capacity and link capacity through another linear problem and again choose a maximal independent configuration set. This process will stop until $K$ configurations are selected or there is no more configuration can be selected.

After the mapping to layers, each node in the layers represents a server with unit capacity, as that in Lemma 2. In step 13, we assign link capacities for nodes that have been assigned to the layers. For each link between nearby layers, the link capacity is assigned with 1 if the corresponding link capacity in the original graph is a positive integer. Otherwise, the link capacity is assigned with 0. In step 14, if there are more server capacity and link capacity available, we assign unit capacity of each server as a node to one layer randomly and assign link capacity if available. Finally, we run the min-cost flow optimization to obtain the integer solution.

Note that our main idea is to use linear optimizations as a basis to select maximal independent configurations for service chains. After that, we use the min-cost flow problem to further optimize the placement. Our approach is novel and can be adapted to more sophisticated optimizations. For example, one can place $K_1 \geq K$ service chains instead of $K$ at the first step. By enlarging the number of service chains to be placed, one can provide more flexibility in configurations and balance the impact of selected configurations and unselected ones. In this paper, we will only adopt Algorithm I, since it has already achieved superior performance, as shown in Section V.

## 4.3 Computation complexity

Here, we show the complexity for Algorithm I.

**Lemma** 5. *The complexity for Algorithm I is polynomial.*

Proof. For Algorithm I, there are two optimizations needed to be solved, i.e., the min-cost flow optimization and the

**Algorithm 1** Function Chain Placement Scheme

```
1: Initialization: $I$ is a vector with $K$ zero elements, the server set in the $l$-th layer $T_l = \emptyset$ $(1 \le l \le L_K)$
2: While $\min(I) = 0$ (some service chains are unselected)
3:    Solve the linear problem $\mathbf{B}$ with the current server capacity and link capacity to place unselected services. Denote the optimal solution as $S = \{C_{k,p_k} | z_{k,p_k} > 0\}$
4:    if $S = \emptyset$, break, go to step 13
5:    for $k = K, K-1, ..., 1$
6:       if $I(k) = 0$
7:          select the highest non-zero utilized configuration $C_{k,p_k} \in S$. If not found, $k = k-1$, go to step 5
8:          $I(k) = 1$. Put the $l$-th server of $C_{k,p_k}$, i.e., $v_l$, in the set $T_l = \{T_l \cup v_l\}$, for $1 \le l \le L_k$
9:          update the server capacity and link capacity, remove the infeasible configurations from $S$
10:      end
11:   end
12: end
13:   Assign link capacities for links between nearby layers
14:   if there are more server capacity and link capacity available, assign the servers to the layers randomly
15: Run a min-cost flow optimization, and record its cost
```

linear problem optimization. The complexity of the min-cost flow is $O(K|V'|^2)$ [14]. We now focus on the complexity of the linear optimization. For theoretical interest, one can use the well known Ellipsoid algorithm, which is of polynomial complexity. For practical interest, the work [15] gives an approximation algorithm for linear problems, where the entries of the constraint matrix are non-negative. Assume the number of non-zero entries in the constraint matrix is $n$, which is $O(KN^L)$ in our case. The algorithm in [15] can run in time $O(n\log(n)/\varepsilon^2)$ to achieve a performance with a gap of $1+\varepsilon$ from the optimality. Therefore, if we assume the parameter $\varepsilon$ as a small constant, the complexity for the linear problem is $O(KN^L\log(KN^L))$.

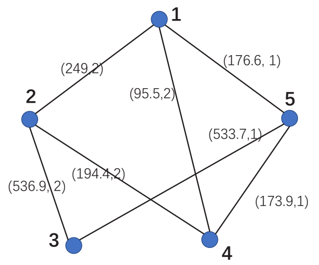
The times to run linear optimization is upper bounded by $K$ and we only run MCFP once. Therefore, the overall complexity of Algorithm I is $O\left(K \cdot KN^L\log(KN^L) + K|V'|^2\right)$. Since $L$ is a constant, we conclude Lemma 5. □

## 5 PERFORMANCE EVALUATION

In this section, we conduct simulations to verify our observations on the properties of the linear problem and the performance of our scheme. We compare our scheme with the greedy scheme, the random rounding scheme and the Kariz scheme proposed in [12].

### 5.1 Case Study

We start by using an example to illustrate how various schemes work to determine the function placement and chaining. In Fig. 3(a), we consider a network composed of five servers. Each server can host up to two functions. When the unit cost on server capacity is assumed to be identical for all servers, Kariz [12] uses a layer-to-layer otpimization. Thus, we assume server costs are also identical. The link cost and capacity between any two servers is labeled on the links between them, i.e., (link cost, link capacity). If two functions are placed in the same server, the link cost is assumed to be zero[4]. In our example, we have two service chains, i.e., service chain 1 requiring three functions and service chain 2 requiring four functions to be placed and chained.



(a) Topology

| Configuration | Utilization |
|---|---|
| (4A,4B,1A,1B) | 0.5 |
| (5A,5B,1A,1B) | 0.5 |
| (4A,5A,5B) | 0.25 |
| (4B,5A,5B) | 0.25 |
| (2A,2B,4A) | 0.25 |
| (2A,2B,4B) | 0.25 |

(b) Optimal fractional solution

**Figure 3: Network topology and the optimal fractional solution.**

**Achieving fractional solutions.** Let us first show how to obtain a fractional solution to the problem. According to Lemma 2, each server $i$ with two units of capacity can be mapped to two servers with unit capacity, denoted as $iA$ and $iB$, respectively. The optimal fractional solution is conducted on this newly constructed graph $G'$, presented in Fig. 3(b). From the figure, we can see that service chain 2 is placed in configurations $(4A, 4B, 1A, 1B)$ and $(5A, 5B, 1A, 1B)$[5], each with a fraction of 1/2. This means the first function in service chain 2 is "divided" into equal two halves, each of which are placed in server $4A$ and server $5A$, respectively. We can also verify that by summing up these partially placed functions, each server hosts up to two functions in total, which satisfies the server capacity constraint. For example, server $1A$ is used in configurations $(4A, 4B, 1A, 1B)$ and $(5A, 5B, 1A, 1B)$, each with a fraction of 1/2. Thus, the sum of the fractions that server $1A$ holds is one. However, in reality, such fractional solution or this "division" on a function is not implementable. We will next show how we achieve integer solutions based on these fractional results.

**Workflow and result of our proposal.** Our scheme maps this fractional solution to a min-cost flow problem. We first identify the maximum independent configuration set. As stated in Section IV, the configuration with the highest utilization (in this example, $(4A, 4B, 1A, 1B)$) is chosen first. Since all other utilized configurations share at least one node with configuration $(4A, 4B, 1A, 1B)$, they cannot be chosen, and the maximum independent configuration set remains a singleton. We then try to place the second service chain with the remaining server capacity and link capacity with a linear programming, from which we will obtain

[4]The cost of self-loop communication can be set as an arbitrary value.
[5]The four functions of service chain 2 are deployed in order in the four servers of the configuration.

configuration $(3A, 5A, 5B)$. We now map the configuration $(4A, 4B, 1A, 1B)$ and configuration $(3A, 5A, 5B)$ into layers, i.e., server $4A$ and server $3A$ into the first layer and server $4B$ into the second layer, etc. Other three servers are mapped randomly to the layers, as illustrated in Fig. 4. Note that the link capacity is also divided. For example, there is only one unit link capacity between server 3 and server 5. Therefore, in the mapped MCFP, the link capacity is assigned to link $(3A, 5A)$ and then can not be used by link $(5A, 3B)$. Now we treat it as a min-cost flow problem and obtain our result, which is shown in the second column of Fig. 5. Note that the cost of configuration $(4A, 4B, 1A, 1B)$ is the summation of link cost $(4A, 4B)$, link cost $(4B, 1A)$ and link cost $(1A, 1B)$, which is $0+242+0 = 242$. Therefore, in our scheme, the configurations selected before MCFP is also $95.5+533.7 = 629.2$. The cost savings by MCFP is then $629.2 - 463.7 = 165.5$.
**Workflow and result of Kariz.** Kariz will first find the minimal cost matching for the first function and second function of two service chains, which can be $(1A, 1B)$ and $(3A, 3B)$. Then Kariz will find the minimal cost matching for the second function (now deployed in $1B$ and $3B$) and third function (can be deployed in the remaining servers) of two service chains, which turns out to be $(1B, 4A)$ and $(3B, 5A)$. Finally, Kariz finds the minimal cost link for the third function and the last function of service chain 2, which is $(5A, 5B)$.

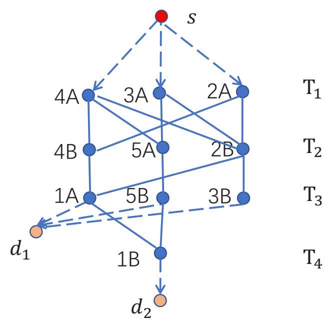


**Figure 4: Mapping to a min-cost flow problem, where real lines represent that the links are valid.**

| Optimal | Our scheme | Greedy | Kariz |
|---|---|---|---|
| (5A,5B,1A,1B) | (2A,4B,1A,1B) | (4A,4B,1A,1B) | (3A,3B,5A,5B) |
| (2A,2B,4A) | (4A,5A,5B) | (3A,5A,5B) | (1A,1B,4A) |
| Cost=371 | Cost=463.7 | Cost=629.2 | Cost=629.2 |

**Figure 5: Solutions for different schemes.**

**Workflow and result of the greedy algorithm.** For comparison, the greedy scheme will choose the configuration with the minimal cost for service chain 2 (with more functions) first, i.e., $(4A, 4B, 1A, 1B)$ in this case, and then the one with the minimal cost from the remaining servers for service chain 1, i.e., $(3A, 5A, 5B)$ in this case.
**Workflow and result of the random rounding algorithm.** For random rounding, it will randomly select $(4A, 4B, 1A, 1B)$ or $(5A, 5B, 1A, 1B)$ for service chian 2, and randomly select one of the four configurations for service chain 1. If the random rounding scheme chooses configurations $(4A, 4B, 1A, 1B)$ and $(4A, 5A, 5B)$, it can achieve a cost of 269.4 even lower than the optimal solution. Unfortunately, the server capacity of server 4 is violated, and this shows the deficiency of using random rounding algorithms.
**Time complexity.** We see that there is a difference in the results given by our scheme and the optimal one. We emphasize by a small increment of total cost, our algorithm significantly reduces the computational complexity. Recall that in Section III, the optimal integer solution is shown to be NP-hard. As the optimal integer solution may only be applicable for small networks, our proposed scheme aims to significantly reduce the complexity by choosing some "favorable" min-cost flow instances instead of all instances. To ensure a desired performance, the properties shown in the relaxed linear problem is of critical importance.

## 5.2 Properties of the linear problem

Now we verify the desired properties for the optimal linear solution obtained in Section 4.1, i.e., the number and utilization of selected configurations can be upper and lower bounded, respectively. We set the simulations without link constraints. The simulation with link constraints is similar. We have $N$ servers, each with one unit capacity. We have $K$ service chains to be implemented, each requiring $L$ functions. The link cost between any two servers is randomly generated from a uniform distribution over $[0, 1000]$.

We run 1000 simulation instances when $N = 26, L = 3, K = 8$ and plot the cumulative distribution of the number of such simulation instances with respect to the performance measures. Figure 6 shows CDF of instances w.r.t. the number of configurations utilized in the optimal fractional solution. We see that over 74% of the instances are with 8 configurations, where the fractional solution is equivalent to the integer solution. The maximal value is 19, which is tighter than the theoretical bound (Lemma 4) $2LK$ (or 48 here).

Recall that in Theorem 2, we show that at least $\frac{K}{2}$ configurations are with utilizations greater than $\frac{1}{4L+1}$ (or 1/13 in this case). Figure 7 shows the CDF of instances w.r.t the minimum utilization of all positive configurations. The lowest value is 0.125, which verifies our theoretic result. In Fig. 8, we show the number of instances w.r.t. the maximal number of disjoint configurations chosen from the fractional solution, where the minimal value is 6. Therefore, if we just adopt these disjoint configurations for the service chain placement, at most two service chains will be rejected. To summarize, Figs. 6-8 verify our theoretic results and show tighter bounds than we proved.

## 5.3 Comparison of placement schemes

Since link capacity constrains may lead to infeasible solutions, we first compare the performance of our scheme with the greedy scheme and Kariz, without capacity constraint. We run 1,000 instances under $N = 26, K = 8, L = 3$. Figure 9 shows the comparison of weighted costs, which refers to the

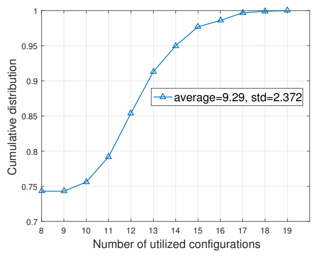


**Figure 6: The number of utilized configurations (LP).**

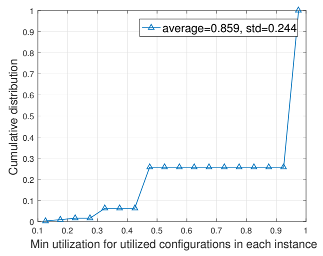


**Figure 7: The minimum utilization for utilized configurations (LP).**

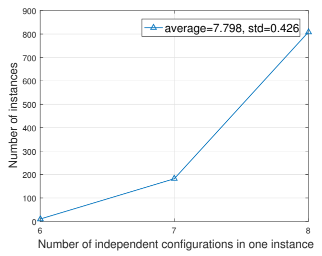


**Figure 8: The maximal number of independent configurations (LP).**

actual cost of the corresponding scheme divided by the minimum cost of the integer solution (exhaustive searching). The $x$-axis is the weighted cost and the $y$-axis is the CDF of number of simulation instances. The weighted cost of the greedy scheme is 1.49 on average, and that of Kariz is 1.83, while that of our proposed scheme is 1.05, which is 44% and 78% lower than greedy scheme and Kariz, respectively. We also note that, when we allow the greedy scheme to choose two best configurations at each time, such as that proposed in [7], it has insignificant performance variation. Figure 10 shows the performance for the random rounding scheme, whose average weighted cost is close to 1. Over 1,000 instances, there are 76% of them whose optimal linear solutions are already optimal integer solutions. For the other 24% simulation instances, 15.3% of them can satisfy the capacity constraints, 82.5% of them have capacity violation of 2, and 2.2% of them have capacity violation up to 3. For comparison, while our scheme has a slight more cost (less than 5%), it satisfies the capacity constraint. These simulation results show the superior performance of our proposed scheme over state-of-art approaches.

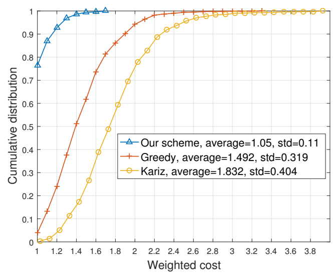


**Figure 9: Comparison of our scheme and other schemes.**

***Remark:*** We note that the superior performance of our scheme comes from two aspects. First, we select maximal independent configurations from a linear problem, which will not saturate the low cost links. In comparison, greedy schemes and Kariz may select a number of good links and leave small room for other configurations or functions to select. Second,

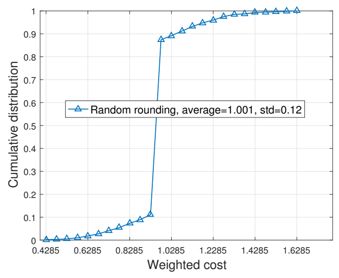


**Figure 10: The performance of random rounding scheme.**

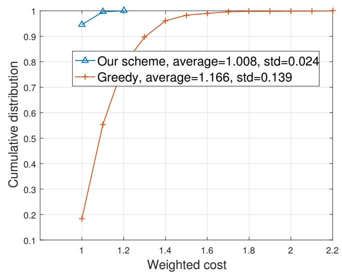


**Figure 11: Performance comparison when services have distinct numbers of functions and there are link constraints.**

by mapping to MCFP, we can further improve the performance, while preserving the integer solution, as shown in our previous example.

Next, we simulate the scenario when the link capacity is taken into account, as presented in Fig. 11. In this example, we have 9 servers, each with two units of capacity. The link capacity between two servers are randomly chosen from $[0, 1, 2]$. Self-loop link capacity is assumed to be zero. There are four service chains in total, one of them having two functions, two of them having three functions each, and the last one having four functions. The greedy scheme first places the service chains with the larger number of functions. Under this setting, our scheme and the greedy scheme always have feasible solutions during the 1,000 simulation instances. We can see that the cost of our proposed scheme is only 1% higher than that of optimal integer solution, which is much better than the greedy scheme.

## 6 CONCLUSION

In this paper, we study the the optimal placement chaining problem for virtual network functions. We set up an optimization model for this integer problem and show underlying properties for its relaxed fractional problem. Based on the properties, we devise an efficient scheme to select low-cost configurations from the fractional solution and determine the mapped min-cost flow problem to obtain the network function placement and chaining solution. Our scheme achieves near-optimal performance and outperforms existing state-of-art approaches.

## ACKNOWLEDGMENT

This work is supported in part by the GRF 14200117 and the Huawei's Research Funding 7010340.

## REFERENCES

[1] European Telecommunications Standards Institute. Network functions virtualization (nfv): Architectural framework. White Paper, 2014.

[2] 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 *Proc. IEEE INFOCOM*, Atlanta, USA, May. 2017.

[3] R. Cohen, L. Lewin-Eytan, J.S. Naor, and D. Raz, "Near Optimal Placement of Virtual Network Functions", in *Proc. IEEE INFOCOM*, Hong Kong, Apr. 2015.

[4] Q. Zhang, Y. Xiao, F. Liu, J. Lui, J. Guo, and T. Wang, "Joint Optimization of Chain Placement and Request Scheduling for Network Function Virtualization", in *Proc. IEEE ICDCS*, Atlanta, USA, June 2017.

[5] M. Sevil, K. Matthias, and K. Holger, "Specifying and Placing Chains of Virtual Network Functions", in *Proc. IEEE CloudNet*, Luxembourg, Oct. 2014.

[6] X. He, T. Guo, E. Nahum, and P. Shenoy, "Placement Strategies for Virtualized Network Functions in a NFaaS Cloud", in *Proc. IEEE HotWeb*, Washington, USA, Oct. 2016.

[7] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood , "Virtual Function Placement and Traffic Steering in Flexible and Dynamic Software Defined Networks", in *Proc. IEEE LANMAN*, Beijing, China, Apr. 2015.

[8] M. Ghaznavi, N. Shahriar, S. Kamaliy, R. Ahmed, and R. Boutaba, "Distributed Service Function Chaining", *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479-2489, Nov. 2017.

[9] G. Even, M. Rost, and S. Schmid, "An Approximation Algorithm for Path Computation and Function Placement in SDNs", *arXiv:1603.09158 [cs.NI]*, Mar. 2016.

[10] M. Rost, and S. Schmid, "Service Chain and Virtual Network Embeddings: Approximations using Randomized Rounding", *arXiv:1604.02180v1 [cs.NI]*, Apr. 2016.

[11] G. Steiner, "On the k-path Partition of Graphs", *Theoretical Computer Science*, vol. 290, no. 3, pp. 2147-2155, 2003.

[12] M. Ghaznavi, N. Shahriar, S. Kamaliy, R. Ahmed, and R. Boutaba, "Distributed Service Function Chaining", *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479-2489, Nov. 2017.

[13] Robert S. Garfinkel, and George L. Nemhauser, "Integer programming", New York: Wiley, 1972.

[14] J. Edmonds, and R. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *Journal of the ACM* , vol. 19, no. 2, pp. 248-264, 1972.

[15] C. Koufogiannakis, and N.E. Young, "A Nearly Linear-Time PTAS for Explicit Fractional Packing and Covering Linear Programs", *Algorithmica*, vol. 70, no. 4, pp. 648-674, 2014.

[16] V. Vazirani, "Approximation Algorithms", Springer, 2003.

## A PROOF OF LEMMA 4

Assume $z$ is an optimal solution to problem $\mathbf{B}'$. Define $I_1 = \{z_{k,p_k} | z_{k,p_k} = 1, z_{k,p_k} \in z\}$, which is the set of configurations whose utilizations are equal to 1. For each configuration in $I_1$, each server and each link in that configuration will be used with one unit capacity. Then we can subtract the used server capacity and link capacity from the original server graph $G$ and obtain a new graph $G_1$.

Define $I_2 = \{z_{k,p_k} | z_{k,p_k} > 0, z_{k,p_k} \in z\}$, which is the subset of non-zero elements in $z$. We have $I_1 \subset I_2$. By subtracting $I_1$ from $I_2$, $I_2 \backslash I_1$ must be an optimal solution for $G_1$. Otherwise, $(I_2 \backslash I_1) \cup I_1$ is not an optimal solution for $G$.

Let $z' = I_2 \backslash I_1$. From the original constraints $Hz \leq b$, we construct a new matrix $H'$ such that $H'z' \leq b'$. For each $z_{k,p_k} \in z$, if $z_{k,p_k}$ is not in $z'$, we remove the corresponding $((k-1)N^L + p_k)$-th column in $H$. We do this for each each $z_{k,p_k} \in z$, and the matrix $H$ is transformed to $H'$. By removing the satisfied service chains in $I_1$ and the corresponding used capacities from $G$, the constraint vector $b$ is transformed to $b'$ in $G_1$. Then, we have $H'z' \leq b'$.

Now we count the number of active inequalities in $H'z' \leq b'$. Note that all $z_{k,p_k} \in z'$ is strictly larger than 0 and smaller than 1. Therefore, the active inequalities in $H'z' \leq b'$ can only appear in the first $K - |I_1|$ constraints, i.e., $\sum_{p_k=1}^{N^L} z_{k,p_k} = 1 (k : \exists z_{k,p_k} \in z')$, and in the constraints for the server capacities and link capacities. As the number of service chains deployed in $z'$ is equal to $K - |I_1|$, the total server capacity needed is no larger than $L(K - |I_1|)$. Note that the server capacity needed in $I_1$ is $L|I_1|$. Therefore, the number of active inequalities in the server capacity constraints of $H'z' \leq b'$ is upper bounded by $\frac{L(K-|I_1|)+L|I_1|}{c_s}$. The total link capacity needed for $z'$ is no larger than $(L-1)(K-|I_1|)$, i.e., there are at most $\frac{(L-1)(K-|I_1|)+(L-1)|I_1|}{c_l}$ active inequalities in the link capacity constraints. Therefore, the total number of active inequalities in $H'z' \leq b'$ is no larger than $K - |I_1| + \frac{LK}{c_s} + \frac{(L-1)K}{c_l}$. By removing the inactive inequalities in $H'z' \leq b'$, $H'$ is transformed to $H''$, and $b'$ is transformed to $b''$, such that $H''z' = b''$. To let $z'$ be an extreme point, $H''$ must have a zero nullspace. Otherwise, assume there exists a $x$ satisfying $H''x = 0$. As every element in $z'$ is strictly positive and smaller than 1, we can choose an arbitrarily small constant $\varepsilon$ such that every element of $z' + \varepsilon \cdot x$ and also $z' - \varepsilon \cdot x$ lies in [0,1]. Therefore, $z'$ is a convex combination of the other two feasible and optimal solutions $z' + \varepsilon \cdot x$ and $z' - \varepsilon \cdot x$, leading to a contradictory.

Since $H''$ has a zero nullspace and the number of rows in $H''$ is not larger than $K - |I_1| + \frac{LK}{c_s} + \frac{(L-1)K}{c_l}$, the number of columns of $H''$ is no larger than $K - |I_1| + \frac{LK}{c_s} + \frac{(L-1)K}{c_l}$. Hence, the number of variables in $z'$ can also be upper bounded, and the number of non-zero elements in the optimal solution $z$ can be upper bounded by

$$K - |I_1| + \frac{LK}{c_s} + \frac{(L-1)K}{c_l} + |I_1|. \tag{14}$$

Thus, we conclude Lemma 4.