A Novel Algorithm for NFV Chain Placement in Edge Computing Environments

Zhiqi Chen[†], Sheng Zhang[†], Can Wang[†], Zhuzhong Qian[†], Mingjun Xiao[§], Jie Wu[‡], Imad Jawhar[♯]
[†]State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
[§]School of Computer Science and Technology, University of Science and Technology of China, P.R. China
[‡]Center for Networked Computing, Temple University, [♯]Midcomp Research Center, Saida, Lebanon
Email: sheng@nju.edu.cn

***Abstract*—Edge computing is gaining popularity these years, more service providers are shifting their services from clouds to the edge for better QoS provision. Recent studies in NFV also tend to deploy Network Function Virtualization (NFV) services in the edge network. However, the NFV deployment in the edge network is a challenging problem and differs from the similar problem in data centers. We mainly focus on a new NFV Chain Placement (NCP) problem in the paper. It is well known that the edge of the network is dynamic, and edge computing aims to utilize the physical edge resources efficiently and quickly. We first prove that the NCP problem is NP-complete. Then we propose a novel metric that can better measure the balance condition of the physical resources. We also analyze its advantages. Finally, we design an efficient algorithm, MINI, based on this metric. We evaluate MINI using extensive simulations. The results show that MINI has great advantages over a genetic algorithm (GA) in terms of physical resource utilization, acceptance rate, and running time.**

***Index Terms*—NFV, Placement, Edge Computing, SDN, Heuristic Algorithm**

I. Introduction

Recently, we observed a considerable amount of trends booming for network domains. Network managers, especially employed in large-scale datacenters, are seeking for more effective and efficient moethods for managing networks. As a result, Network Function Virtualization (NFV) [1], [2] has emerged as a new approach to designing, deploying, and managing network infrastructure. It decouples the network functions from proprietary hardware and runs them as software applications on general purpose hardware. This shift in paradigm toward "softwarization" allows cost reduction and service agility. Traditionally, middle-boxes widely deployed in datacenters are proprietary hardware devices. However, the substantial dependence of networks on their underlying hardware and the existence of various specialized hardware appliances, such as firewalls (FW), deep packet inspection (DPI) equipment, and network address translation (NAT) in the network infrastructure have escalated the challenges facing network service providers [3]. NFV provides many benefits to the telecommunications industry. Some of these benefits are openness of platforms, scalability and flexibility, operating performance improvement, and shorter development cycles.

In NFV infrastructure, hardware-based devices are no longer needed. Instead, we merely take use of general commodity servers located in small cloud nodes which are distributed across the entire network. Concretely, network functions would be virtualized and operated on general commercial machines. For instance, a service chain is often composed of one or more network functions like IDS, Firewall, and VPN.

As we know, most related studies have focused on NFV Placement problem in cloud computing environments, which is different from our work. Driven by the visions of Internet of Things and 5G communications, recent times have seen a shift in computing paradigm, from the centralized Cloud Computing towards Edge Computing (EC) [4], [5]. The main feature of EC is to push mobile computing, network control and storage to the network edges (e.g., base stations and access points) so as to enable computation-intensive and latency-critical applications at the resource-limited edge devices. Edge computing promises dramatic reduction in latency, tackling the key challenges for materializing 5G vision. The promised gains of EC have motivated extensive efforts in both academia and industry on developing the technology. This paper mainly focuses on the NCP problem in the edge computing environments. We believe that a perfect scheme of NFV placement could decrease unnecessary wastes of resources.

In this paper, we propose a feasible and brand-new criteria to evaluate the efficiency of NCP scheme in edge computing environments and demonstrate that the NCP problem is NP-Complete. Afterwards, this problem is formulated as an Integer Nonlinear Programming (INLP) problem, and we propose a heuristic algorithm to solve it. To evaluate our algorithm, a genetic-based algorithm (GA) is designed for comparison.

The rest of the paper is organized as follows. In section II, we present the related works. In section III, we give the formal definitions of the NCP problem in edge environments and a novel performance metric is introduced. In section IV, we prove that the NCP problem is NP-Complete. In section V, we propose and analyze our heuristic algorithm. In section VI, we evaluate our algorithm by simulations. We conclude this paper in Section VII.

II. Related Work

Previous works in NFV mainly focus on how to deploy network functions on commodity servers and to minimize the total placement cost [6], [7], [8]. In [6], the authors aim to minimize the distance between clients and virtual network functions to provide better QoS. Meanwhile, the setup cost of these functions is taken into consideration. In [9], the authors study the problem of VNF Placement with replications and proposed a GA heuristics to solve this problem. The other related work [10] focuses on the VNF Managers (VNFM) placement and aims at minimizing the operational cost without violating the performance requirements. In addition, recent researchers [11] consider NFV Placement as an Integer Linear Problem (ILP) and approximation algorithm based on ILP relaxation and rounding have been proposed. Another work [12] is proposed for placement of multi-component applications in edge computing environments , which is related to our work. Roughly speaking, they model the user application as an application graph and the physical computing system as a physical graph. However, in our paper, NCP problem in edge network is a variant of Bin Packing Problem [13], which is quite different compared to previous works. The Bin Packing Problem was investigated widely in [14], [15].

III. Problem Formulation

A. Overview

We introduce two kinds of graphs called Application Graph (G_A) and Physical Graph (G_P). A mapping from G_A to G_P corresponds to a feasible NFV placement scheme. The essence of the problem is how to construct an injective mapping from G_A to G_P. For convenience, we introduce some notations which would be used later.

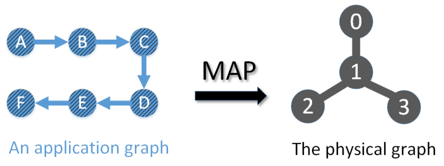

Fig. 1. One application graph is mapped onto the physical graph.

B. Definitions

1) Definition of Application Graph: When network traffic comes, it is supposed to go through the designated service chain in the right order. In this paper, we consider network functions as application graph nodes and suppose these service chains are in a linear shape, which is assumed in many works [3], [8], [9]. An application graph can be denoted as $G_A = (V_A, E_A)$, where each vetex corresponds to a NF node in the service chain. The weight of each node is calculated by a function $c\text{-}weight(\cdot)$, which represents its demands for computing resources, storage resources, and bandwidth in real scenarios. Analogously, the weight of an edge in the graph indicates the required network bandwidth for the two nodes it connects. For instance, suppose the bandwidth between NF function $v_A(i)$ and NF function $v_A(j)$ is at least 100 kbps, $b\text{-}weight(i,j)$ should be set to 100. Table I shows the notations which would be used for an application graph.

2) Definition of Physical Graph: The physical device network could be represented in a physical graph as well, which is denoted as G_P. Actually, nodes and edges on the physical graph represent physical devices and physical links deployed in Software-defined networking (SDN). Suppose

TABLE I
The Notations for the Application Graph

Symbols	Definitions
$G_A^k = (V_A^k, E_A^k)$	The kth application graph for the kth service chain
$V_A^k = V(G_A^k)$	The vertex set of the kth application graph
$v_A^k(i) \in V_A^k$	The ith vertex of the kth application graph
$c\text{-}weight(i)$	The amount of resources demanded by vertex $v_A^k(i)$
$E_A^k = E(G_A^k)$	The edge set of the kth application graph
$e_A^k \in E_A^k$	One of the edges of the kth application graph
$e_A^k(i,j)$	The vertexes $v_A^k(i)$ and $v_A^k(j)$ are connected by the edge e_A^k
$b\text{-}weight(i,j)$	The essential amount of bandwidth required by the edge $e_A^k(i,j)$

$G_P = (V_P, E_P)$ (denote the physical vertex set as V_P and the physical edge set as E_P). For each vertex $v_P \in V_P$, let $c\text{-}capacity(i)$ and $b\text{-}capacity(i,j)$ denote the maximum computing capacity of the physical node $v_P(i)$ and the maximum bandwidth capacity of connections between node $v_P(i)$ and node $v_P(j)$. Table II shows the notations related to the physical graph G_P.

TABLE II
The Notations for the Physical Graph

Symbols	Definitions
$G_P = (V_P, E_P)$	The physical graph
$V_P = V(G_P)$	The vertex set of the physical graph
$v_P(i) \in V_P$	The ith vertex of the physical graph
$c\text{-}capacity(i)$	The amount of resources which can be provided by vertex $v_P(i)$
$E_P = E(G_P)$	The edge set of the physical graph
$e_P \in E_P$	One of the edges in the edge set of the physical graph
$e_P(i,j)$	The vertexes $v_P(i)$ and $v_P(j)$are connected by the edge e_P
$b\text{-}capacity(i,j)$	The amount of bandwidth resources which can be provided by the physical edge $e_P(i,j)$

3) Definition of Mapping π: As we mentioned before, a mapping π from G_A to G_P corresponds to a feasible NFV placement scheme, since each function in the NFV chain should be assigned to a server node in G_P. Additionally, we use an indicator variable $x_{ij}^k \in \{0,1\}$ to indicate whether $v_A^k(i)$ is deposed at $v_P(j)$. As a matter of fact, the following constraint shoule be obeyed:

$$\sum_j x_{ij}^k = 1, \forall i, k \tag{1}$$

As for constraints on edges, we introduce an indicator variable $y_{pqst}^k \in \{0,1\}$ to indicate whether $e_A^k(p,q)$ goes through $e_P(s,t)$, resulting in the following constraint:

$$\sum_s \sum_t y_{pqst}^k \geq 1, \forall k, p, q \tag{2}$$

C. Optimization Goal and Constraints

Maximizing the resource utilization ratio is an essential objective of NFV placement, so we aim to reduce the resource fragmentation on physical servers after the placement of NFVs. However, there are several issues worth our attention:

1) How to evaluate the utilization: Fairly it is unsuitable to directly use the whole utilization ratio of G_P (the ratio of used physical capacity to overall physical capacity) as our optimization objective, because no matter how G_A are placed, the utilization ratio of G_P will never change. To solve the problem above, we redesign our objective function. We decide to maximize **the sum of the squares of the remaining capacity in all physical nodes**, which is a more challenging problem. The appropriateness behind the new metric is that the capacity of each physical node is limited, and we should make full use of these capacities, so the remaining capacity of each physical node should be as small as possible after the application nodes are deployed. We could impose a "penalty" factor on the remaining capacity of each node (but accumulating the remaining capacity of each node directly would cause the same problem as the former method, because essentially the two methods are the same in mathematical sense). By introducing the sum of the squares of the remaining capacity, we could avoid the annoying problem and better know about the condition of the network resource allocation.

2) Whether bandwidth utilization should be taken into consideration in the optimization objective: The answer is no. As far as we know, more than one application graph edge could be mapped onto the same physical graph edge at the same time, which is a disadvantage for considering bandwidth utilization. If our algorithm was aimed to maximize the bandwidth utilization of G_P, it would be easily inclined to a terrible solution which triggers great waste of bandwidth. The algorithm would attempt to separate two nodes for the purpose of maxmizing the bandwidth utilization.

3) How to set computing and bandwidth constraints: On the one hand, for each $v_P(j)$, the sum of the resources required by all $v_A^k(i)$ deployed above denoted as $\sum_k \sum_i x_{ij}^k c\text{-}weight(k,i)$ cannot exceed the capacity of $v_P(j)$, which is denoted as $c\text{-}capacity(j)$.

$$\sum_k \sum_i x_{ij}^k c\text{-}weight(k,i) \le c\text{-}capacity(j), \forall j \tag{3}$$

On the other hand, each $e_A^k(p,q)$ in G_A^k corresponds to multiple edges in G_P after mapping from G_A^k to G_P. This suggests that for each $e_P(s,t)$, the sum of the bandwidth required by the corresponding edge of G_A^k that it carries cannot exceed the maximum actual bandwidth that $e_P(s,t)$ could provide. i.e., $\forall s,t$, we have

$$\sum_k \sum_p \sum_q y_{pqst}^k b\text{-}weight(k,p,q) \le b\text{-}capacity(s,t) \tag{4}$$

where the left part is the bandwidth required and the right part is the bandwidth capacity.

D. Optimization Objective

The optimization objective is to maximize the sum of the squares of the remaining capacity in all physical nodes. The formulation is given as follows (constraints are based on former statements):

$$\begin{aligned} \max_{\pi} \quad & \sum_j \left(c\text{-}capacity(j) - \sum_k \sum_i x_{ij}^k c\text{-}weight(k,i) \right)^2 \\ \text{s.t.} \quad & \\ & \sum_k \sum_i x_{ij}^k c\text{-}weight(k,i) \le c\text{-}capacity(j) \quad \forall j, \\ & \sum_k \sum_p \sum_q y_{pqst}^k b\text{-}weight(k,p,q) \le b\text{-}capacity(s,t) \quad \forall s,t, \\ & \sum_j x_{ij}^k = 1 \quad \forall k,i, \\ & \sum_s \sum_t y_{pqst}^k \ge 1 \quad \forall k,p,q, \\ & x_{ij}^k \in \{0,1\} \quad \forall k,i,j, \\ & y_{pqst}^k \in \{0,1\} \quad \forall k,p,q,s,t \end{aligned} \tag{5}$$

IV. NP-Completeness of NFV Placement Problem

We can prove that the NCP problem is NP-Complete by reducing Bin Packing Problem to it. Bin Packing problem is well-known as a NPC problem. The proof is as follows.

- Decision version of Bin Packing Problem: Given n items $\{a_1, \cdots, a_n\}$, N bins (each with capacity V),and $\mathbb{B}$ (whose capacity is B), the question is whether we can use no more than B bins to hold all items.
- Decision version of NCP Problem: Given n G_As, G_P and T, the information related to the two types of graphs such as weights of nodes and edges is all available (the weight of v_A are noted as $\{c_1, \cdots, c_n\}$ and the capacity of v_P are noted as $\{s_1, \cdots, s_{|G_P|}\}$). The question is whether there exists a mapping π according to which we can figure out a NFVs placement scheme whose objective value (calculated by (5)) is at least T.

When given a mapping π from G_A to G_P, we are able to verify whether the sum of squares of remaining capacity can reach T or not by simulation. It is easily concluded that the complexity of the verifying operation is polynomial. Hence, the NCP problem belongs to **NP**.

Next, we remove the constraint on capacity of edges and suppose that all n G_A merely contain a single node (their computing resources are notated as $\{c_1, \cdots, c_n\}$). The reduction from Bin Packing Problem to NCP problem is as follows:

- $\{\mathbb{B}_1, \cdots, \mathbb{B}_N\} = G_P$
- $a_i = c_i, \forall i \in \{1, \cdots, n = |G_A|\}$
- $V = s_1 = \cdots = s_{|G_P|}$

If we have an algorithm which can figure out the optimal solution π_{OPT} for Bin Packing Problem in polynomial time, then the solution of NCP problem could be constructed accordingly. We would next prove that the solution we obtain is also optimal in NCP problem.

Denote the operation of moving an item from one bin to another as $\mathcal{T}$, based on which the current placement plan π is transformed into a new placement plan $\pi^{'}$ as : $\pi \xrightarrow{\mathcal{T}} \pi^{'}$.

***Theorem** 4.1:* In an operation $\mathcal{T}$, if an empty box is involved, $T_{new} > T$ (T_{new} is the cost of new placement scheme) would increase.

Proof. Consider two bins $\mathbb{B}_1, \mathbb{B}_2$ and suppose there is an item with weight x_1 in $\mathbb{B}_1$ before the transformation. Then assume that the total weight of other items is in B1 is x_2 and $\mathbb{B}_2$ is empty. As we mentioned before T is the sum of squares of remaining capacity, in current placement we have $T = (V - x_1 - x_2)^2 + V^2$. After the movement, we have can update T to T_{new} which can be expressed as $T_{new} = (V - x_2)^2 + (V - x_1)^2$. The difference between T and T_{new} turns out as $2x_1x_2 > 0$, which means $T_{new} > T$. Therefore, theorem 4.1 follows immediately. $\square$

***Theorem** 4.2:* The NCP problem is NP-Complete.

Proof. Assuming π_{OPT} is not optimal in the NCP problem, denote $\pi^{'}_{OPT}$ as optimal. Due to optimality of π_{OPT}, $\pi^{'}_{OPT}$ might occupy more bins than π_{OPT} (according to Theorem 4.1), T of π_{OPT} is less than that of $\pi^{'}_{OPT}$, which violates optimality of $\pi^{'}_{OPT}$ in NCP. In summary, π_{OPT} is optimal in the NCP problem. $\square$

V. Solution

Since we are informed that NCP problem is NP-Complete, if we attempted to solve it optimally, it should take tremendous computing resources and operating time inevitably, which is against our expectations. Hence, an efficient heuristics algorithm is designed. In the following section, we will introduce our algorithm with fast operating time and good performance.

A. MINI Algorithm

Algorithm 1 Minimal Neighbourhood (MINI) Algorithm

```
Input: $G_P, G_A^k (k = 1, \dots, K)$
Output: $place$
1:  init $place$
2:  for $k = 1$ to $K$ do
3:      $parent \leftarrow$ null
4:      for node $i$ in $G_A^k$ do
5:          $candidates \leftarrow$ seek for $parent$ or $parent$'s neighbours if it's a satisfied node
6:          if $candidates == \varnothing$ then
7:              $candidates \leftarrow$ do BFS until find one candidate
8:          end if
9:          place node $i$ at the minimal remaining c-capacity one in $candidates$
10:         update $place$, $parent$, all capacity information
11:     end for
12: end for
13: return $place$
```

In this section, we propose a greedy algorithm called Minimal Neighbourhood (MINI) Algorithm. The principle idea of MINI is that when an new G_A arrives, MINI would attempt to find a v_P with the least remaining computing capacity (but v_P has enough capacity to accommodate the first G_A node). As for the rest G_A nodes, following principles should be obeyed:

- If possible, deploy the current node at *parent* node (the v_P where the last v_A is deployed).
- If the *parent* node overflows, turn to seek for the neighbors of *parent*, which satisfy computing and bandwidth constraints as *candidates*.

For the next step, we would choose the v_P with minimal c-capacity from *candidates* as the goal node; if *candidates* are empty, we have no choice but make a breadth-first search (BFS). This would loop until it discovers the v_P which satisfies the constraints. Then we update bandwidth constraint information along the returned paths. However, if BFS fails, our MINI algorithm has to claim that there is no solution found. We can find that the algorithm complexity of MINI is $O(K \times |V_A| \times (|V_P| \log |V_P| + |E_P|))$. MINI has several advantages as follows:

- Putting things into the smallest bucket is a greedy strategy which seems simple but works very well in solving the NCP problem. Actually, the solution of it approximates the optimal solution.
- The principle of proximity placement makes the bandwidth requirement for the entire application graph much less, leaving enough remaining bandwidth capacity for subsequent nodes.
- This heuristic strategy is fast and has good performance. Besides, it is also very easy to be developed into an online algorithm, because the only thing we need to do is to use the algorithm to perform corresponding calculations whenever a new G_A arrives.

B. Example

Let us give an example to better understand the operation of the algorithm. As Figure 2 shows, suppose we have G_P and 3 G_A to be placed, whose sequences are determined by order of arrival. According to the MINI algorithm, the first node $v_A^1(\mathbf{A})$ in G_A^1 should be placed at the minimal node $v_P(4)$ in G_P, then the remaining capacity of $v_P(4)$ should be 0. Fig.2 shows the case that G_A^1 is placed onto G_P completely.

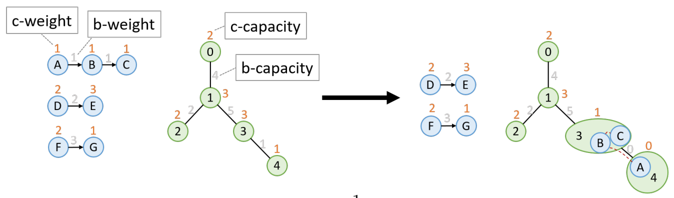

Fig. 2. Place G_A^1 onto G_P.

Next, we choose to place G_A^2. Since its first node capacity is 2 (although the node capacity of $v_P(4)$ and $v_P(3)$ is smaller at the same time, the placement constraints are not satisfied), we can only place it on $v_P(2)$, and then treat it as *parent* to find the placement of subsequent nodes in the future.

Finally, when we are placing the last application graph G_A^3, we can only choose to place $G_A^3(\mathbf{F})$ to $G_P(0)$ (for the reason

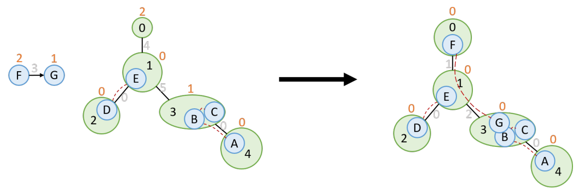

Fig. 3. Place G_A^3 onto G_P

of constraints). As a result, $parent$ is $G_P(0)$. However, because the $candidates$ which satisfy the placement constraints cannot be found, the MINI algorithm chooses to perform a BFS and finds the suitable node $v_P(3)$. Consequently, all 3 G_A are placed, and resources of all these physical graph nodes are used up luckily.

C. Baseline : Genetic Algorithm

In order to demonstrate efficiency of the heuristic algorithm, we compare the MINI algorithm with the GA algorithm [9] in simulation.

Algorithm 2 Genetic Algorithm (GA)

```
Input: $Population, Generation, TargetFitness$
Output: $\vec{x}$
 1: Pop $\leftarrow$ Generate 1st generation randomly
 2: for $i = 1$ to $Generation$ do
 3:   newPop $\leftarrow \varnothing$
 4:   for $p = 1$ to $Population$ do
 5:     if any Pop's fitness $\geq TargetFitness$ then
 6:       Break;
 7:     end if
 8:     Pick up two individuals from Pop by fitness
 9:     if random(0,1) $<$ Probabilty of crossover then
10:       Cross two individuals
11:     end if
12:     if random(0,1) $<$ Probabilty of mutation then
13:       Mutate two individuals
14:     end if
15:     newPop $\leftarrow$ newPop $\cup$ two new individuals
16:   end for
17: end for
```

Genetic Algorithm is a meta-heuristic method targeted at providing an approximate optimal solution for general optimization problems. In the NCP problem, the solution space contains all mappings from V_A to V_P. Define the quantity of V_A as m and the quantity of V_P as n, as well as the size of its solution space is n^m. In fact, we could represent a problem solution as a string. Concretely, consider a string S, whose length is the same as that of G_A. S_i stands for v_P holding $v_A^k(i)$, such as the physical node A, B or C. The string is similar to the form $AAB \cdots AC$. In addition, it is better to design Fitness Function $fitness$ in advance (with the constraints of nodes' and edges' capacity). The Crossover Function $crossover$ and the Mutation Function $mutation$ would be utilized to the gene exchange and the gene mutation among strings in one generation.

VI. Evaluation

Firstly, we study a case in which the application graphs and the physical graph are shown in Figure 1. The weight of nodes in the physical graph is set according to array $\{200; 0; 200; 300\}$, and the weight of edges and nodes in G_A are both generated according to a uniform distribution. Afterwards, we use our MINI algorithm to solve the NCP problem, and the GA-based algorithm is complemented for comparison. The square term of the remaining empty space is a metric to evaluate quality of placement scheme generated by these two algorithms.

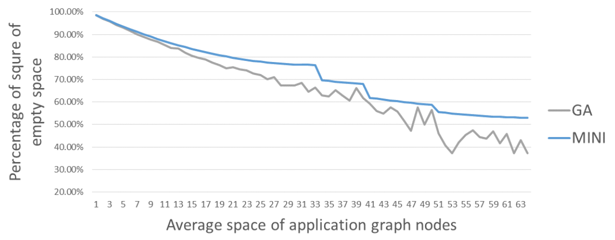

Fig. 4. With the increase of the mean computing weight of application graphs, the percentage of square of remaining empty space decreases in GA and MINI.

From Figure 4, we can see that with the increase of the mean of nodes in application graphs, the cost of the two algorithms gradually decreases, indicating that the remaining capacity on servers is gradually occupied by NFV nodes. Among them, the fitted curve of MINI Algorithm is smoother , which shows how the MINI algorithm outperforms the GA algorithm in stability.

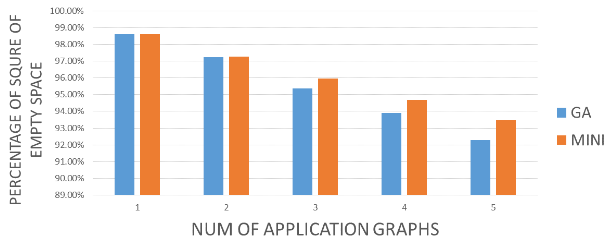

Fig. 5. As the number of application graphs increases, the cost calculated by the two placement algorithms has begun to decline.

Then, we set up a controlled experiment as follows. To study the influence of the number of G_A on the algorithm performance, we increase the number of application graphs while keeping the quantity of physical nodes unchanged. The GA and MINI algorithm was exploited to solve each test cases separately. As we can see from Figure 5, as the number of application graphs increases, the cost calculated by the two placement algorithms declines, but the solutions generated by MINI are better than that of the GA algorithm. At the same time, with the increase of the number of application graphs, the difference of two algorithms becomes more significant. It is illustrated in Fig.5. that the solution generated by the MINI algorithm is better than GA algorithm in general cases with respect to the solution quality.

Figure 6 shows the running time of these two algorithms. Apparently, MINI algorithm use much less time to generate

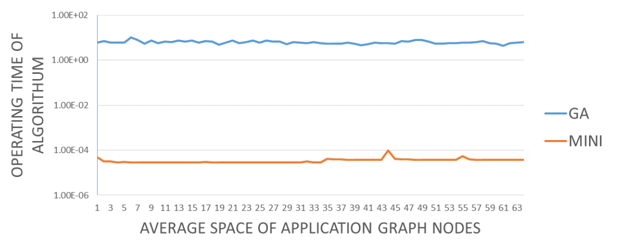

Fig. 6. The average operating time of GA algorithm and MINI algorithm.

the final results than the GA algorithm, it can still figure out the results within milliseconds even when the size of application graphs grow larger. In the mean time, we can see that the size of application graphs does not have a great impact on the operating time of the two Algorithms. The fast speed of MINI algorithm makes it suitable and practical for the edge computing environments. In conclusion, MINI achieves a better tradeoff between optimality and running time.

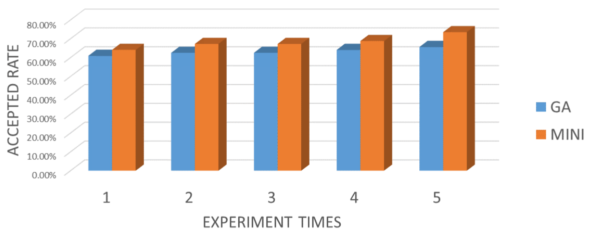

Fig. 7. The average accepted rate of GA algorithm and MINI algorithm.

We simulate the arrival of K application graphs randomly, and then use the two algorithms to calculate the allocation cost of application graphs separately. In the whole process, we perform the simulation 64 times and count the number of times that they are successfully accepted. From Figure 7 we can find that the acceptance rate of the two algorithms is about 63%(GA) and 68%(MINI), indicating that the performance of MINI in acceptance rates is superior to that of GA.

The above work uses a simple simulation of the program. Conducting extensive simulations using real-world traces is part of our future work.

VII. Conclusion

Network Function Virtualization (NFV) has emerged as a new approach to designing, deploying and managing network infrastructure. It decouples the network functions from proprietary hardware and runs them as software applications on general purpose hardware. This shift in paradigm toward "softwarization" allows cost reduction and service agility. Previous works in NFV mainly focus on the placement of NFVs in data centers; different to them, we rethink the NCP problem at network edge [16]. We first propose a new metric to evaluate the efficiency of the NCP problem. Its main idea is to reduce the capacity fragmentation caused by the deployment of NFV as much as possible, and thus the resources of physical devices can be fully utilized. Secondly, we have proved that the NCP problem is NP-Complete by reducing Bin Packing Problem to it. Thirdly, we propose a heuristic algorithm called MINI to solve the NCP problem. In the evaluation, we compare MINI with a GA-based approch and discover MINI has significant improvements in processing time, service acceptance rate, and resource utilization rate compared to GA. The fast speed of the MINI algorithm makes it suitable and practical for the edge computing environments. Besides, MINI is flexible, since it can achieve a tradeoff between optimality and running time. We believe that our work can help network managers to orchestrate NFV efficiently.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2017YFB1001801), NSFC (61502224, 61472181, 61472185), CCF-Tencent Open Fund, Ministry of Education & China Mobile Research Foundation (MCM20170307), NSF (CNS 1757533, CNS 1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and IIP 1439672), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[3] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[4] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1346–1354.

[7] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[8] W. Ma, J. Beltran, Z. Pan, D. Pan, and N. Pissinou, "Sdn-based traffic aware placement of nfv middleboxes," *IEEE TNSM*, vol. 14, no. 3, pp. 528–542, 2017.

[9] F. Carpio, S. Dhahri, and A. Jukan, "Vnf placement with replication for loac balancing in nfv networks," in *Proc. of ICC 2017*. IEEE, 2017, pp. 1–6.

[10] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. W. Tchouati, "On the placement of vnf managers in large-scale and distributed nfv systems," *IEEE TNSM*, vol. 14, no. 4, pp. 875–889, 2017.

[11] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi, and W. Kellerer, "Qos-driven function placement reducing expenditures in nfv deployments," in *Proc. of ICC 2017*. IEEE, 2017, pp. 1–7.

[12] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[13] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[14] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 312–320.

[15] S. S. Seiden, "On the online bin packing problem," *Journal of the ACM (JACM)*, vol. 49, no. 5, pp. 640–671, 2002.

[16] C. Wang, S. Zhang, H. Zhang, Z. Qian, and S. Lu, "Edge cloud capacity allocation for low delay computing on mobile devices," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE, 2017, pp. 1–8.