# Evolvable Virtual Network Function Placement Method: Mechanism and Performance Evaluation

Mari Otokura *Student Member, IEEE*, Kenji Leibnitz, Yuki Koizumi *Member, IEEE*, Daichi Kominami *Member, IEEE*, Tetsuya Shimokawa, Masayuki Murata, *Member, IEEE*

***Abstract*—In Network Functions Virtualization (NFV), network functions are operated in software as Virtual Network Functions (VNFs) instead of dedicated hardware. The most important issues that need to be addressed in NFV are where the VNFs should be placed in the network, as well as what amount of resources should be assigned to each VNF. *Evolvable VNF Placement* (*EvoVNFP*) is a meta-algorithm that we previously proposed for controlling an underlying iterative VNF placement method. EvoVNFP realizes better adaptability to regular demand changes by mimicking biological evolution under time-varying environments leading to faster generation of placements. We provide detailed evaluation studies about the mechanism of EvoVNFP and show that iterative placement methods combined with EvoVNFP can generate placements that adapt better to varying goals because of *triggers*. Numerical results verify that EvoVNFP is able to reduce the required number of calculation steps by up to 48%.**

***Index Terms*—Network functions virtualization, software defined networks, evolution, modularly varying goals.**

## I. Introduction

Communication services have continuously evolved to become more diverse and dynamic. Considering the ongoing realization of the *Internet of Things* (IoT), these trends are expected to become even more distinct in the future. Previously, communication service providers have been utilizing network functions, such as firewalls or intrusion detection systems, implemented in hardware to provide their services. However, it is becoming more and more difficult to deal with the ongoing diversification of communication services through hardware solutions because this requires large capital expenditure (CAPEX) and operating expenditure (OPEX) to add and maintain new communication services.

*Network Functions Virtualization (NFV)* [1], [2], [3] is a promising technology for dealing with this situation. The basic idea of NFV is to separate the network functions from their physical computational resources, such as CPU, memory, or storage. These network functions are implemented in software running on *virtual machines* (VMs) as *Virtual Network Functions* (VNFs). VMs are usually run on commodity-type physical computational resources. Unlike its hardware counterpart, NFV enables to change the performance of network functions dynamically at runtime. Furthermore, NFV also provides the pipelining of virtual network functions referred to as *Service Function Chaining* (SFC) [4], [5].

Recently, many researchers have approached the problem of placing VNFs on physical servers according to specific objectives [6], [7], [8], [9], [10], [11]. Furthermore, the ETSI NFV ISG specification [12] suggests that VNF chains can be broken down into smaller parts than functions, which would lead to an even higher complexity of the placement problem.

A challenging point is that the VNF placement problem needs to be solved in a dynamic manner when considering SFC to deal with dynamic request changes from users [13], [14], [15], [16], [17]. While placements of NFV in a network should be optimized according to requirements of the users, they also need to be updated whenever VNFs arrive, depart, or change in configuration. Furthermore, the time needed to calculate new placements should be kept as short as possible in order to provide continuous services. An intuitive way of dealing with this problem is by solving this optimization problem every time there is an arrival, departure, or change of requests. However, this may become too computationally intensive, since already the static VNF placement problem itself is NP-hard [11].

In order to tackle this problem, we utilize knowledge from biological evolution: when biological organisms evolve towards two (or more) varying environments, they inherently reach a system that is highly adaptable to both of these environments. Kashtan *et al.* [18] proposed a meta-algorithm called *Modularly Varying Goals* (MVG), which controls an underlying *genetic algorithm* (GA) so that it adapts to varying environments by utilizing the concept of biological evolution. GAs are algorithms that evolve a population of individuals (*genomes*) over several generations in order to fit a certain (usually single) goal and they are commonly used as heuristics for various optimization problems. Furthermore, the same authors showed in [18] that MVG forms genome structures that are highly adaptable to varying environments, which can also help to speed up evolution [19].

In previous work [20], [21], we introduced the concept of MVG to the VNF placement problem by proposing a meta-algorithm called *Evolvable VNF Placement* (EvoVNFP) operating above any iterative VNF placement methods. The underlying placement method combined with EvoVNFP does not reinitialize its current solution whenever the situation changes due to a new arrival, departure, or change of requests. Furthermore, our method enforces periodical changes of its

Manuscript received January XX, 2018; revised YY, 2018.

M. Otokura, Y. Koizumi, and M. Murata are with the Graduate School of Information Science and Technology, Osaka University, 1-5 Yamadaoka, Suita, Osaka 565-0871, Japan.

K. Leibnitz and T. Shimokawa are with the Center for Information and Neural Networks (CiNet), National Institute of Information and Communications Technology (NICT) and Osaka University, 1-4 Yamadaoka, Suita, Osaka 565-0871, Japan.

D. Kominami is with the Graduate School of Economics, Osaka University, 1-7 Machikaneyama, Toyonaka, Osaka 560-0043, Japan.

goals every fixed number of generations in order to optimize its structure. These features of EvoVNFP help the placement method to find fast solutions that are robust to environmental changes. Similar to [22], we refer to this as the evolvability of the network and therefore name our proposal EvoVNFP.

In [20], [21], we evaluated the performance of EvoVNFP operating in combination with a GA. Important issues remain, though, which we address in this current paper. First, we have only evaluated EvoVNFP with GA and not with other iterative placement heuristics. Second, we have considered only a rather small and unrealistic network environment consisting of five routers and ten physical machines. Third, we have not discussed in detail why the placements generated by EvoVNFP are robust. In this paper, we discuss the reasons of the higher adaptability of EvoVNFP from the structure of its solutions with information-theoretic metrics [23] on larger networks than in our previous studies [20], [21]. We also compare the results of state-of-the-art VNF placement methods [16], [17] operating with and without EvoVNFP. We found that methods combined with EvoVNFP can configure placements that have a small number of important elements in individuals needed for adapting the dynamics, referred to as *triggers* in [23]. We showed the existence of the triggers by evaluating conditional entropy and mutual information. We also compare the performance of conventional iterative optimization methods with those enhanced by EvoVNFP and show that the number of calculation steps to converge is reduced.

This paper is organized as follows. We first discuss previous work related to this study in Sec. II. Section III provides the system model as well as the explanation of the VNF placement problem and its solutions. We then introduce the control algorithm EvoVNFP in Sec. IV and analyze the structure of the solutions that are generated by a placement method combined with EvoVNFP in Sec. V. In Sec. VI, we show the results of a comparative evaluation of EvoVNFP with two state-of-the-art VNF placement methods from [16], [17]. Section VII concludes this paper.

## II. Related Work

*Virtual Network Embedding* (VNE) [24], [25], [26] considers a similar problem to that of VNF placement by embedding *Virtual Networks* (VNs) consisting of virtual nodes and virtual links into substrate networks. Similarly to VNFs, VNs are also operated in software and can therefore utilize network resources in a more flexible and efficient way than hardware solutions. However, there is one major difference between VNE and the VNF placement problem: While in VNE users request VNs that consist of multipoint-to-multipoint network connection requests, they request Service Chains (SCs) with point-to-point flow routing demands when dealing with the VNF placement problem [9].

Several papers have addressed the VNF placement problem from the viewpoint of system modeling. For example, Moens et al. [6] proposed and evaluated an Integer Linear Programming (ILP) model of the VNF placement problem to minimize resource consumption in a hybrid VNF scenario, where network functions are provided in either hardware or software. Similarly, to minimize resource consumption, Gupta et al. [7] proposed four kinds of models which differ in the degree of flexibility in the deployment of service chains. Cho et al. [8] proposed a model based on real measurements to capture network latency among VNFs. Additionally, several papers [9], [10] considered multi-objective optimization by minimizing the maximum link utilization, number of utilized resources, or consumed bandwidth.

Cohen et al. [11] argued that the VNF placement problem is NP-hard because it includes two well-known NP-hard optimization problems: the facility location problem and the generalized assignment problem. Furthermore, the demands can change dynamically in the real world. Therefore, it is required to calculate and configure placements rapidly. While a few of these papers directly approach the ILP with solvers, e.g., CPLEX, most of the approaches addressing the dynamic VNF placement problem use approximation algorithms or heuristic methods to perform fast calculations of the solution. Zhang et al. [13] provided a model via graph pattern matching and an approximation algorithm, while Liu et al. [14] proposed a heuristic algorithm for finding near-optimal solutions with low computational complexity, and Bari et al. [15] provided a dynamic-programming based heuristic. Metaheuristics are also frequently used for quickly finding solutions of the dynamic placement problem. For instance, Rankothge et al. [16] regarded the dynamic VNF placement as a combination of initial placement and scaling of VNFs, and proposed two respective algorithms based on GA to find a good placement. Furthermore, Mijumbi et al. [17] addressed the problem of dynamic VNF placement and scheduling, and proposed an algorithm based on tabu-search as well as three variants of the greedy algorithm.

In summary, most of the aforementioned studies tackle the VNF placement problem by mathematical models or heuristics. In contrast, our proposal EvoVNFP is a meta-algorithm that operates as an overlay above any iterative VNF placement heuristic. By iteratively changing the objective function, EvoVNFP is able to find placement solutions that are more adaptable to future traffic changes.

## III. System and Problem Overview

In this section, we explain the system considered in this study as well as the VNF placement problem and its solutions.

### A. Overview of System

An overview of our assumed system is illustrated in Fig. 1. This system consists of several physical machines (PM$i$), each accommodating one or more virtual machines (VM$j$). The physical machines are interconnected via routers $r_k$ and links to form a physical network topology as shown at the bottom of Fig. 1. Furthermore, the system has a controller which computes the placements of VNFs and deploys them to the physical system. When the controller receives a new request for a VNF chain from a user (Step 1), it decomposes each VNF in the chain into multiple functional *components* (Step 2) [12]. For example, a firewall can be split into two components: one

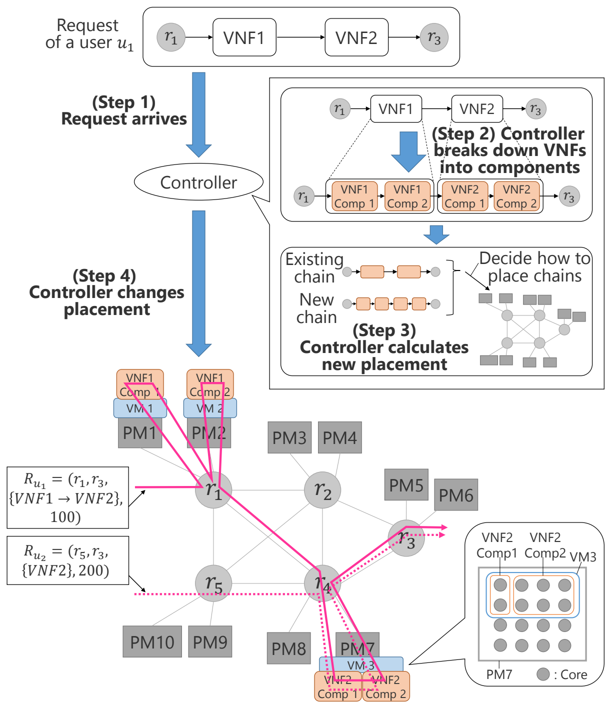

Request of a user $u_1$
$r_1$
VNF1
VNF2
$r_3$
(Step 1) Request arrives
Controller
(Step 2) Controller breaks down VNFs into components
VNF1 Comp 1
VNF1 Comp 2
VNF2 Comp 1
VNF2 Comp 2
Existing chain
New chain
Decide how to place chains
(Step 3) Controller calculates new placement
(Step 4) Controller changes placement
VM 1
VM 2
VM 3
PM1
PM2
PM3
PM4
PM5
PM6
PM7
PM8
PM9
PM10
$r_2$
$r_4$
$r_5$
$R_{u_1} = (r_1, r_3, \{VNF1 \rightarrow VNF2\}, 100)$
$R_{u_2} = (r_5, r_3, \{VNF2\}, 200)$
VNF2 Comp1
VNF2 Comp2
● : Core

Fig. 1. An operational overview of the system. When a new request for a VNF chain arrives (Step 1), the controller breaks down the VNF into functional components (Step 2), determines their optimal placement (Step 3), and finally assigns this placement to the physical system (Step 4).

for classification of packets and another for dropping illegal packets. This decomposition of VNFs enables the grouping of functional parts that can be shared by multiple VNFs, leading to a better utilization of the physical resources. Next, the controller calculates a new placement of the VNFs and VMs that satisfies all chains in the system, i.e., it solves the VNF placement problem (Step 3). Finally, the controller deploys the new placement to the physical system (Step 4). The main focus of this paper lies on the calculation of placements at the controller in Step 3.

## *B. Problem Description and Solution Method of VNF Placement*

In this subsection, we provide an overview of the VNF placement problem and explain how solutions are found for these problems. The input to the VNF placement method is the information about all VNF chains that are currently being requested by users. The output is the allocation (placement) of the virtual resources to the physical network. In the VNF placement problem, components occupy CPU cores on the VMs they are assigned to that in turn occupy PMs. The more cores are used for VMs and components, the better their performance becomes. The number of cores assigned to components should be set to adequately process the traffic passing through them. Furthermore, the number of cores for components and VMs must not exceed the physical limits of their accommodating VMs and PMs, respectively. Note that a single VM can host multiple components if it has a sufficient number of cores to accommodate them. Likewise, a single PM can also accommodate multiple VMs. The objective of the VNF placement method is to generate a placement which meets predetermined criteria, e.g., minimizing the number of used PMs, while also obeying the constraints, e.g., not exceeding the capacity of a PM. We will explain the models and methods that will be used for the evaluation in detail in Sec. V.

## *C. Iterative Solution Methods*

In this subsection, we discuss about iterative methods that keep improving the solution until it reaches a sufficient quality. Most of such iterative methods are metaheuristics. We hereafter focus on two iterative methods that have been proposed for solving the VNF placement problem, a genetic algorithm and tabu search.

*1) Placement with a Genetic Algorithm (GA):* First, we describe the VNF placement with a genetic algorithm. GA imitates biological evolution in a highly distributed manner and operates on a population of possible solutions (*individuals*) to the given optimization problem encoded as a bit-string (*genome*). The evaluation of genomes is performed each generation on the basis of a *fitness function*, which the GA iteratively keeps trying to improve. Over the course of several generations, all genomes in the population with the exception of an *elite* set are randomly modified by genetic operators, mutation and crossover, and new genomes are being produced while old genomes that are less suitable for survival are eliminated in the next generation. One advantage of GAs is that although they are in essence a random search over the search space, the quality of the best solution keeps improving with the number of generations. The typical sequence of a GA is as follows:

1) **Initialization:** Initialize the population of individuals as a set of solution candidates.
2) **Fitness calculation:** Calculate the fitness of each individual as the value representing how well it fits the environment using a predetermined fitness function.
3) **Mutation and crossover:** Randomly modify the individuals by mutation and crossover. *Mutations* change only small parts of an individual, while *crossovers* combine two individuals to form new offspring with traits from both of their parents.
4) **Selection:** Select good individuals from the population with the highest fitness for reproduction to form the new population of the next generation.
5) **Termination:** Decide whether the algorithm should terminate or not according to a predetermined criterion. If this criterion is not yet met, return to Step 2.

The loop composed of Steps 2–5 is referred to as one *generation*. While repeating multiple generations, the fitness of all individuals in the population will gradually improve. The final individuals usually do not reach the globally optimal solution, but a sufficiently good suboptimal result that is usually sufficient for real applications. One advantage of GA is its calculation speed since it can calculate solutions in near real time while many other optimization methods require much longer computation time. Note that in the basic GA algorithm shown above Step 5 is a termination step, while our GA for the

**Algorithm 1** Genetic Algorithm [16]

```
1: $P \leftarrow$ Initialize()
2: while StoppingCriteriaNotSatisfied() do
3:     $F \leftarrow$ CalculateFitness($P$, $f$)
4:     $P \leftarrow$ Selection($P$, $F$)
5:     $P_m \leftarrow$ Mutation($P$)
6:     $P \leftarrow$ GenerateNewPopulation($P$, $P_m$)
7: end while
8: $p \leftarrow$ SelectBestIndividual($P$)
Output: $p$
```

dynamic VNF placement problem continues with its execution forever without terminating.

The algorithm of GA is shown in Algorithm 1 [16]. After initialization of population $P$ (line 1), the following calculation continues until the stopping criteria are satisfied (line 2). First, fitness $F$ is calculated from $P$ and an evaluation function $f$ as the scores for all individuals in the population (line 3) and the individuals in $P$ are arranged according to their corresponding scores (line 4). Then, the individuals in $P$ are mutated and saved as $P_m$ (line 5). The mutation changes one of the partial placements in the selected full placement and consists of two operations: replacing the VNFs to other PMs and rewiring the paths of the chains. This GA only utilizes mutations and no crossovers. Finally, a new population $P$ is generated from the old $P$ and $P_m$ (line 6). After the stopping criteria has been satisfied, the best individual in $P$ is selected as the output (line 8).

*2) Placement with Tabu Search:* In *tabu search*, we continuously generate a set of solutions which are close to the current solution, called *neighborhood*, and select the best solution in the neighborhood. We also save the operation which is used to generate the next solution from the current solution in a *tabu list* for several steps. This prevents us from repeating the operations in the tabu list, which helps preventing getting stuck in locally optimal solutions. In tabu search, it is possible to choose a worse solution than the current one as the next solution, and therefore we separately save the best solution so far as the best solution.

An algorithm of tabu search [17] is shown in Algorithm 2. The current solution $z$, the best solution $z^*$, and a tabu list $T_l$ are initialized at the beginning (line 1), after which the following loop continues until stopping criteria are satisfied (line 2). First, $N(z)$ is calculated as the neighborhood of $z$, while being careful not to do operations in $T_l$ (line 3). Then, the best solution in $N(z)$ is selected as $z$ and $T_l$ is updated (line 4–5). If the score of $z$ is higher than that of $z^*$, $z^*$ is replaced with $z$ and $T_l$ is updated (line 6–9). After the stopping criteria has been satisfied, $z^*$ is selected as the output.

## IV. Proposed Meta-Algorithm for Dynamically Controlling VNF Placements

In this section, we explain the algorithm of our proposal EvoVNFP, which is a meta-algorithm operating above an iterative VNF placement method. First, we provide an overview of how EvoVNFP controls the VNF placement method. Second, we summarize the concepts of EvoVNFP and its underlying method MVG [18] in Sec. IV-B. Third, we explain the mechanism of EvoVNFP in greater detail.

**Algorithm 2** Tabu Search [17]

```
1: $z, z^*, T_l \leftarrow$ Initialize()
2: while StoppingCriteriaNotSatisfied() do
3:     $N(z) \leftarrow$ CalcNeighborhood($z$, $T_l$)
4:     $z \leftarrow$ CalcBestIndividualInNeighborhood($N(z)$)
5:     $T_l \leftarrow$ UpdateTabuList($T_l$)
6:     if IsCurrentScoreHigherThanBestScore($z$, $z^*$) then
7:         $T_l \leftarrow$ UpdateTabuList($T_l$)
8:         $z^* \leftarrow z$
9:     end if
10: end while
Output: $z^*$
```

### A. Overview of EvoVNFP

EvoVNFP has two distinct features. We will explain them by using Fig. 2. This figure shows how a pure method, e.g., GA or tabu search, and a method with EvoVNFP run when computing their solutions over time. EvoVNFP controls goals of an underlying method, as shown in the figure. The first feature of EvoVNFP is that it periodically switches between two (or more) evaluation functions every fixed number of steps, referred to as *period*, which calculate the goodness of the placements, even when there is no external change of the traffic. In Fig. 2, the pure method uses a single evaluation objective function at a certain epoch between two sequential changes of requests while the method with EvoVNFP uses two kinds of evaluation functions, the original objective function and a relaxed objective function, and switches between them every period. The relaxed objective function is created by artificially modifying the original objective function. The second feature of EvoVNFP is its reuse of candidate solutions. The pure methods generally make new candidates of solutions at the beginning of each epoch. However, the methods with EvoVNFP utilize the final candidates of solutions from the evaluation functions of the previous epoch as the initial candidates of solutions for the subsequent evaluation functions.

Next, we will explain details of the concepts of MVG and EvoVNFP. These two concepts differ in their behavior and therefore their resulting structures are also different.

### B. Mechanism of Modularly Varying Goals (MVG)

Our proposal EvoVNFP is inspired by *Modularly Varying Goals* (MVG) [18], which is a meta-algorithm for controlling a GA to obtain a modular grouping of individuals. Instead of computing the fitness for a single goal like the pure GA, MVG imitates biological evolution under varying environments by switching between multiple goals every fixed number of generations. If the underlying problem is "modular", i.e., it can be decomposed into basic functions that deal with shared subfunctions contributing to the current goal, MVG is able to find these common modules. For example, [18] presents an example for the construction of a circuit according

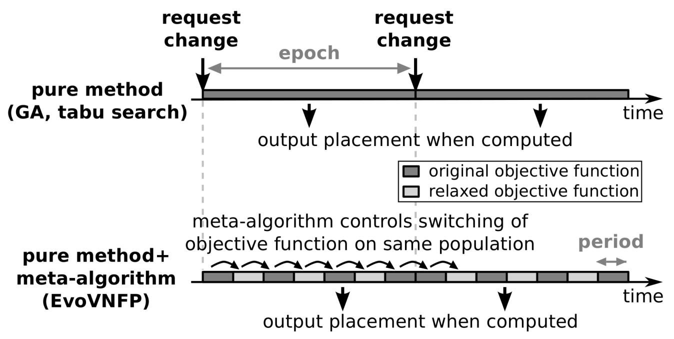

request change
request change
epoch
pure method (GA, tabu search)
time
output placement when computed
original objective function
relaxed objective function
meta-algorithm controls switching of objective function on same population
period
pure method+ meta-algorithm (EvoVNFP)
time
output placement when computed

Fig. 2. An overview of a pure method (top) and a method with EvoVNFP (bottom)

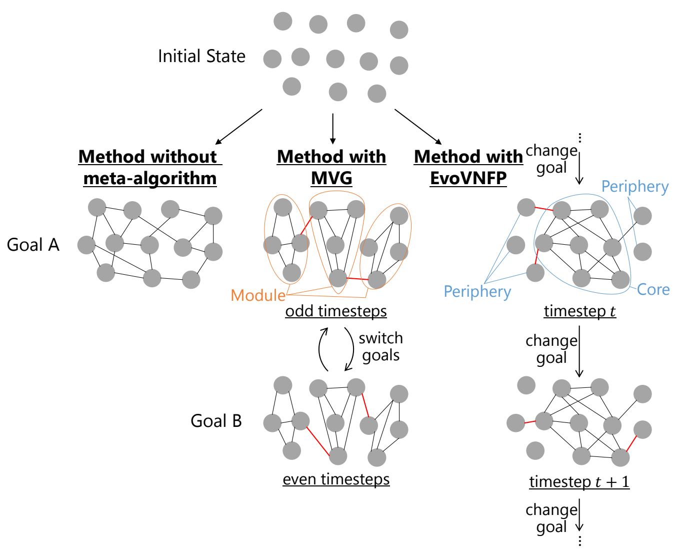


Fig. 3. Differences in concepts of changing evaluation functions and their impact on the resulting topology type.

to given logical functions (goals) and where the nodes are logical gates. The authors designed a logic circuit which has four boolean input variables $X, Y, Z,$ and $W$, and a single output with the GA iteratively switching between two logical functions over time as goals $G_1 = (X \oplus Y) \wedge (Z \oplus W)$ and $G_2 = (X \oplus Y) \vee (Z \oplus W)$. In this case, the basic functional modules that $G_1$ and $G_2$ have in common are $(X \oplus Y)$ and $(Z \oplus W)$. Due to construction, both $G_1$ and $G_2$ are composed of the same two basic function modules, but combine these two modules with different operators ($\wedge$ and $\vee$) for $G_1$ and $G_2$, respectively. As a result, MVG eventually generates solutions consisting of the basic function modules that do not change over time and the respective operators linking the modules. This permits an efficient and fast switching between goals $G_1$ and $G_2$ that only involves the change of a single operator. By this way, MVG is able to control GAs to generate solutions with a modular structure that correspond to the modularity of the goals.

Furthermore, it was shown in [19] that MVG can generate their solutions quickly as the iterative switching between goals decreases the probability that solutions get stuck at local optima. A pure GA uses only a single goal and therefore its evolution may eventually stagnate. On the other hand, MVG switches between at least two goals, which modifies the direction of the search in the solution space of the evolution process. Note that a certain improvement could also be achieved if a GA would switch between arbitrary non-modular goals, but it was shown in [19] that the highest speedup in convergence is achieved if the goals are indeed modular.

### C. Evolvable VNF Placement (EvoVNFP)

We outline the basic mechanism of our proposal *Evolvable VNF placement* (EvoVNFP) [20], [21], which applies the concept of MVG to the VNF placement problem.

The mechanisms of EvoVNFP explained in Sec. IV-A assist in generating solutions with a (temporal) "core-periphery" rather than a modular structure in the placements. Modularity and core-periphery are two distinct categories of network topologies that have been well studied. Modular networks have dense connections between nodes of the same module and sparse connections between nodes in different modules [27]. On the other hand, a core-periphery structure has one (or more) densely connected groups of nodes, referred to as *core*, which remains stable over time even when goals are varied, and a *periphery* of sparsely connected nodes which change more rapidly in order to adapt to the goal changes [28]. This structure helps reduce the costs for reconfiguring placements whenever requests arrive or depart, while maintaining the chains for the remaining requests. The concepts of a pure method without meta-algorithm, a pure method with MVG, and a pure method with EvoVNFP are illustrated in Fig. 3. The resulting network structure obtained from the method without meta-algorithm would be rather random while the method with MVG, and the method with EvoVNFP produce more regular structures. However, the latter two differ in their regularity. While the method with MVG produces a more modular structure, the method with EvoVNFP generates a core-periphery structure. In the dynamic VNF placement problem, the core-periphery structure reduces the time to calculate subsequent placements because the placement can be easily reconfigured by only considering their differences (periphery). This also reduces the reconfiguration costs of VM placements and component placements, and the amount of resources allocated to them in the physical network.

What remains is for us to describe how EvoVNFP switches the evaluation functions periodically between actual evaluation functions (dark gray periods in Fig. 2) and relaxed ones (light gray periods in Fig. 2). Relaxing an evaluation function means that we do not consider the originally requested chain in its entirety as goal for our heuristic, but only a subset of its functional modules. This behavior is inspired by an example of MVG for evolving RNA structure in [19]. When relaxing an evaluation function, it should remain mostly similar to the original evaluation function because if the two varying goals do not have any common parts, our method would not be able to find a good modular overlap.

We show a psuedocode of EvoVNFP in Algorithm 3. The input parameter is the length of periods $T_p$. First, we initialize an original goal $g_o$ and make a set of relaxed goals $G_r$ from

**Algorithm 3** Algorithm of EvoVNFP

```
Input: $T_p$
 1: $g_o \leftarrow$ InitializeGoal()
 2: $G_r \leftarrow$ GenerateRelaxedGoals($g_o$)
 3: $l \leftarrow 0$
 4: while StoppingCriteriaNotSatisfied() do
 5:     if GoalChangedAtPreviousStep() then
 6:         $g_o \leftarrow$ UpdateGoal()
 7:         $G_r \leftarrow$ GenerateRelaxedGoals($g_o$)
 8:     end if
 9:     if ($l$ mod $T_p$) = 0 then
10:         $g_c \leftarrow$ SwitchCurrentGoal($g_o$, $G_r$)
11:     end if
12:     DoOperationsOfIterativeMethod($g_c$)
13:     $l \leftarrow l + 1$
14: end while
```

$g_o$ (line 1 and 2). Then the underlying iterative method starts running. EvoVNFP checks for two things at the beginning of every step. One is whether new arrivals, departures, or changes of requests occurred at the previous step or not (line 5). If they occurred, we update $g_o$ and calculate its new $G_r$ (line 6 and 7). The other check is whether the current step is when we must switch the goal, i.e., whether the equation ($l$ mod $T_p$) = 0 is met or not, where $l$ is the number of total elapsed steps (line 9). If such a timing takes place, we switch the goal and save it as a current goal $g_c$ (line 10). After EvoVNFP has finished these checks, the underlying method calculates placements using $g_c$ (line 12). The underlying method stops when the stopping criteria are satisfied (line 4) and finally the underlying method outputs a placement as a solution.

## V. Numerical Evaluation of EvoVNFP Mechanism

We now investigate the performance of EvoVNFP. In order to study the adaptability of EvoVNFP, we first investigate the quality of the solutions from EvoVNFP with a simple GA. We first explain the model and the problem formulation of the VNF placement problem from [20], the settings used for the simulation runs, and the evaluation metrics, followed by the numerical results and their discussions.

### A. Model of NFV System

To relate the number of cores of a component with its performance, we evaluate the performance of one core by its *processing amount*, i.e., the number of instructions it can process, and define the term *processing capacity* as the maximum processing amount that cores can process per time unit (seconds). We assume that the performance of a PM and VM are proportional to the number of cores they can use. Then, the processing capacity of a machine with $n$ cores is $n \cdot C$, where we define $C$ as the processing capacity of a core.

We define *request flows* as flows of the traffic of the users who send *requests* to the controller. A request $R_u$ from user $u$ is represented as $R_u = (src_u, dst_u, chain_u, b_u)$: $src_u$ is the ingress router, $dst_u$ is the egress router, $chain_u$ is the chain of VNFs, and $b_u$ is the requested transmission rate. Figure 1 shows an example of two request flows $R_{u_1}$ and $R_{u_2}$ of users $u_1$ and $u_2$. Note that VNF components can be shared if multiple chains of requests include the same VNFs.

We assume that queuing delay occurs in both routers and components in this study. The queuing delay is not constant because it depends on the current performance and load of the components. The propagation delay, on the other hand, is assumed as a constant delay when the request flow passes through a particular link. We define *request flow delay* $t_u$ as the sum of all queuing delays and all propagation delays of the request flow from user $u$.

In order to consider the queuing delays analytically, we have to define arrival and service rates at routers and components. Service rate of router $r$ is assumed as a constant value $M_r$ packet/s and service rate of component $j$ is $\mu_{k,j,a} = (n_{k,j,a} \cdot C)/T_a$ packet/s, where $n_{k,j,a}$ is the number of cores which the component $j$ of VNF $a$ occupies on VM $k$ and $T_a$ is the processing amount needed by VMs which have components of VNF $a$ to process a packet. Arrival rate at router $r$ is $\lambda_r = \sum_{r'} p^u_{(r,r')} \cdot b_u$ bit/s, where $\lambda_r$ is the sum of the transmission rates of all request flows arriving at the router $r$ and $p^u_{(r,r')}$ is a binary indicator variable which is 1 when the request flow of a user $u$ passes through a link between routers $r$ and $r'$, and otherwise 0. Arrival rate at component $j$ is the sum of the transmission rates of all the request flows which request VNF $a$ and it is represented as $v_a$ bit/s. For the sake of simplicity, we assume that each component behaves like an M/M/1 queuing system.

The delay of the entire placement is calculated after the paths of the request flows have been decided. The delays of each traffic flow are first calculated and then they are averaged with weights corresponding to the transmission rates. Finally, the total delay of a placement $\hat{d}$ is calculated as the average of request flow delays $t_u$ weighted with the transmission rates $b_u$ of each user $u$ as:

$$\hat{d} = \sum_u \frac{b_u}{\sum_u b_u} \cdot t_u.$$

### B. Formulation of VNF Placement Problem

We showed in [20] that the VNF placement problem can be described by Eqs. (1)–(4) based on the model explained in Sec. V-A.

$$\begin{aligned}
\text{minimize} \quad & \hat{d} + W \cdot \sum_{i,k} m_{i,k} && && (1)\\
\text{subject to} \quad & L_a \cdot \frac{v_a}{S} \le n_{k,j,a} \cdot C && \forall k, j, a && (2)\\
& \sum_k m_{i,k} \le N_i && \forall i && (3)\\
& \sum_{j,a} n_{k,j,a} \le m_{i,k} && \forall k, i && (4)\\
\text{variables} \quad & m_{i,k},\ n_{k,j,a}
\end{aligned}$$

Definitions of the symbols used in Eqs. (1)–(4) are given in Table I. Our objective is to reduce the average delays for all chains in the system and the total number of used CPU cores

TABLE I
DEFINITIONS OF SYMBOLS USED IN VNF PLACEMENT PROBLEM

| | |
|---|---|
| $\hat{d}$ | Average delay of all chains in the system |
| $W$ | Weight coefficient |
| $m_{i,k}$ | Number of cores that VM $k$ occupies on PM $i$ |
| $L_a$ | Processing amount that VMs need to provide to components of VNF $a$ |
| $v_a$ | Arrival rate of traffic at VNF $a$ |
| $S$ | Size of one packet |
| $n_{k,j,a}$ | Number of cores that component $a$ of VNF $j$ occupies on VM $k$ |
| $C$ | Maximum number of instructions that a CPU core can process per unit time (s) |
| $N_i$ | Number of cores of PM $i$ |

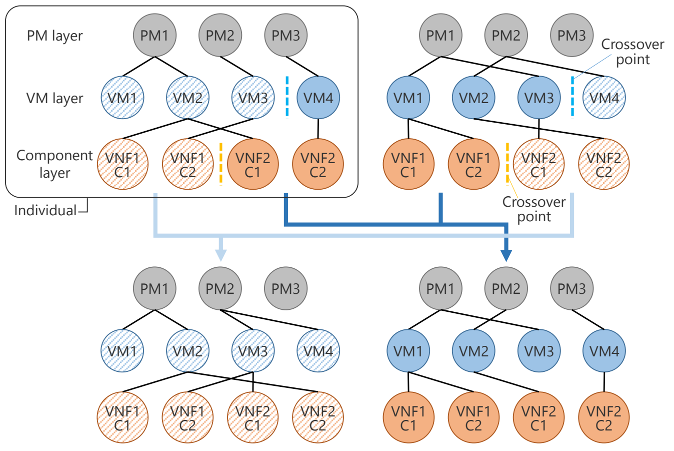


Fig. 4. Example of crossover in GA between two individuals representing VNF placements

as shown in Eq. (1). Constraint (2) means that there must be a sufficient number of cores for a component to process all traffic passing through it. Constraint (3) means that the number of cores for all VMs on a PM must not exceed the total number of cores of the PM. Likewise, Constraint (4) means that the total number of cores for all components on a VM must not exceed the number of available cores of the VM.

## C. Design of GA Parameters

We explain the design of GA used in this subsection for the analysis of the mechanisms of EvoVNFP.

*1) Individuals:* We specifically design the individuals for application to this VNF placement problem. In Fig. 4, four simplified schematics of individuals are shown. Each individual has 3 layers: a PM layer, a VM layer, and a component layer which corresponds to the three elements in the VNF placement model: PMs, VMs, and components, respectively. Connections between the PM layer and the VM layer represent assignments of VMs to PMs and connections between the VM layer and component layer represent placements of components on VMs. Furthermore, nodes maintain the information of the number of cores that each corresponding PM, VM, or components has.

*2) Fitness function:* We design the fitness function as

$$F = \left( \frac{\hat{d}}{d_{max}} + \frac{W(\sum_{i,k} m_{i,k})}{c_{max}} \right)^{-1} \quad (5)$$

if the individual meets the constraints given in Eqs. (2)–(4) of Sec. V. Here, $d_{max}$ is the delay of a request flow with the maximum number of components, each component having 80% utilization, and the hop length being the maximum hop length in the physical network plus three, and $c_{max}$ is the maximum possible number of cores.

If the individual violates any one of the constraints, we simply set fitness as $F = \alpha Z$ where $\alpha$ is a negative constant, e.g. $\alpha = -0.1$, and $Z$ is the number of violations against the constraints.

*3) Crossovers and mutations:* Crossovers and mutations are operations that randomly change the individuals in a similar way as in real biological evolution. A crossover consists of randomly selecting two individuals, setting crossover points in each of the VM and component layers, and recombining the left and right parts crosswise from the two parent genomes to two new child genomes, see Fig. 4. Each node in the VM and component layers has only one link directed to its upper layer (PM and VM, respectively). The two individuals at the top of the figure are the original ones prior to the crossover. The crossover points on the VM and component layers are marked as blue and orange dashed lines. The two individuals at the bottom of this figure are the ones which are the result of this crossover.

Mutations are performed by randomly selecting one of the following four operations:

- choose an existing link between the PM and VM layers and replace the PM node with a different node on the same layer,
- choose an existing link between the VM and component layers and replace the VM node with a different node on the same layer,
- change the number of cores of a node on VM layer, or
- change the number of cores of a node on component layer.

*4) Selection:* After calculating the return values of the fitness functions, GA ranks all solutions in descending order of their return values. A fixed number of high-ranking individuals is saved as *elites* and directly passed to the next generation without crossovers or mutations. All other lower-ranking individuals must undergo mutations and crossovers as described above before being passed to the next generation.

## D. Simulation Settings

We consider a physical network composed of 40 routers where each router is connected to 2 PMs, i.e., there are 80 PMs in total, and each PM has 32 CPU cores. The physical topology of this considered network is assumed to be that of the pan-European data network for the research and education community GÉANT [29]. The propagation delay on the physical links between any two routers is 20 ms.

For our evaluation, we assume a discrete-time system with the time units of generations. Intervals of arrivals and sojourn times of the requests in the system are variable and follow geometric distributions because we assume that this system is a standard queuing system. We used three kinds of arrival rates: high (1 request per 333 generations), middle (1 request per 500 generations), and low (1 request per 1000 generations). The higher the arrival rates, the more difficult the problem in assigning resources becomes. Furthermore, the patterns of the request arrivals and departures are the same for GA with and without EvoVNFP. We assume that there are 4 kinds of VNFs called VNF1, VNF2, VNF3, and VNF4 that can be arbitrarily combined into chains. Users request one of the 40 possible chains among all combinations of VNF1–4 that have a maximum length of 3.

All other parameters are chosen based on realistic values, such as currently available commercial machines or switches[30], and they are summarized in the following. The requested transmission rate $b_u(t)$ of user $u$ is 50 Mbit/s, the processing capacity of a single core is $C = 3.0$ GHz, the size of a packet $S = 1500$ bit, and the service rate of router $r$ is $M_r = 3.0$ Gpacket/s.

Updates for improving the real physical placements are only performed if the newly computed solution is at least 10% better than the current real placement. We assume that there are 20 requests already in the system at the beginning of the simulation in order to reduce the transient period. The maximum number of requests is set to 40 and we reject any requests that would exceed this number.

The number of generations for each simulation run is 10000, the total number of individuals is 1000, of which there are 100 elites. Probabilities for mutation and crossover are 0.8 and 0.5, respectively. The individuals of the GAs have 80 nodes in their PM layers, which corresponds to the number of PMs in the physical network. They also have 320 nodes in their VM and component layers, respectively. The length of the periods of the fluctuating goals $T_p$ in EvoVNFP is set to 10 generations.

## E. Evaluation Metrics

The metrics we use in the following evaluation fall into two categories. In the first category, we evaluate the performance of the GA and GA with EvoVNFP from the viewpoint of the VNF placement problem. The three considered metrics in this category are as follows.

*1) Success probability:* The probability that the methods can obtain feasible solutions between any two situation changes. It is calculated from all results of a simulation run.

*2) Delay and number of cores:* We also evaluate the quality of generated placements in terms of average *delay* of all chains in the best placements over all generations and the *number of cores* in the best placements in all generations. We calculate the results of the two metrics at every generation.

*3) Costs for reconfiguration:* We evaluate the *costs for reconfiguration* needed to update a placement [31]. These costs should be kept small to reduce the downtime of the services and consist of two metrics: the number of migrations and the number of resizings. A migration means that a VM or a component is moved from one PM to another PM, whereas resizing means that the number of cores assigned to a VM or a component is changed. Here, we only consider stateless network functions, which can be arbitrarily migrated.

The second category includes five evaluation metrics for the internal structure of all solutions (individuals) in the population.

*1) The number of used PMs:* The number of PMs which accommodate one or more VMs. This metric represents how widely the components are distributed over the physical network. The larger this metric is, the more distributed the components are.

*2) The number of used VMs:* The number of VMs that accommodate one or more components. This metric represents how much the chains requesting the same components share these components among each other. The smaller the results of this metric are, the more the components are shared.

*3) Genetic variation of each position in individuals:* The entropy of the values of each position [23]. In this evaluation, the positions correspond to the elements in the individuals, i.e., nodes and links, and the entropy $H(X_i)$ for position $i$ is defined in Eq. (6),

$$H(X_i) = -\sum_j \beta_i \log \beta_i \tag{6}$$

with $X_i$ representing a random variable of the value in position $i$, $\beta_i = \sum_k \eta_i P(T = T_k)$, and $\eta_i = P(X_i = j|T = T_k)$. The larger $H(X_i)$ is, the more diverse the values in position $i$ are among all individuals in the population during the whole simulation. Given that there are for example three individuals in the population and if VM1 of the three individuals in the population occupies 10 cores, the $H(X_i)$ of VM1 is minimal. On the other hand, if VM1 of the three individuals occupies 5, 7, and 9 cores, respectively, the $H(X_i)$ of VM1 is maximal.

*4) Conditional entropy $H(X_i|T)$ and mutual information $I(X_i, T)$:* These metrics for the values of each position of the goals are defined in Eqs. (7) and (8), respectively,

$$H(X_i|T) = -\sum_k H(X_i|T = T_k) \tag{7}$$

$$I(X_i, T) = H(X_i) - H(X_i|T) \tag{8}$$

where $T$ represents a random variable of states of goals and $H(X_i|T = T_k) = -\sum_j P(X_i = j|T = T_k) \log P(X_i = j|T = T_k)$. The term $H(X_i|T)$ represents how diverse the values at position $i$ are when the population evolves towards each goal, and $I(X_i, T)$ becomes large when the values at position $i$ are diverse during a whole simulation but not diverse among the populations which are evolving towards each goal. They are used to detect the positions which are important for adapting to new goals, called *triggers* in [23]. These triggers are positions which have low $H(X_i|T)$ and high $I(X_i, T)$, which means that the values of these positions change frequently in the generations when goals change but do not change frequently when goals do not change. The authors of [23] showed that these positions appear clearly in MVG but not in normal GA which only uses a single goal to evolve.

Figure 5 represents an example of triggers in a VNF placement. The individual on the right represents the placement on the left. Comp1 and VM1 in the placement, i.e., the nodes which represent Comp1 and VM1 in the individual, are

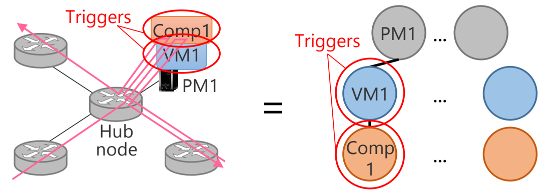


Fig. 5. Example of triggers in VNF placement problem

triggers because changing the number of cores for them affects almost all chains which pass through the hub node in the physical network. Moreover, Fig. 6 provides an explanation for the detection of triggers by calculating entropy $H(X_i)$, conditional entropy $H(X_i|T)$, and mutual information $I(X_i, T)$. $H(X_i)$ represents how the values of position $i$ among individuals in the population are diverse in the simulation. The larger $H(X_i)$ is, the more diverse they are. On the other hand, $H(X_i|T)$ represents how each set of values corresponding to each goal of position $i$ are diverse. The larger $H(X_i|T)$ is, the more diverse each set of values are. According to Eq. (8), when $H(X_i)$ is large and $H(X_i|T)$ is small, $I(X_i|T)$ becomes large. Large $I(X_i|T)$ means that the values at position $i$ are significantly modified at goal changes, but not diverse when the goal is kept. We use both of $H(X_i|T)$ and $I(X_i|T)$ to detect the trigger because high $I(X_i|T)$ does not always mean that $H(X_i|T)$ is low. $I(X_i|T)$ becomes high when both of $H(X_i)$ and $H(X_i|T)$ are high and when both are low.

## *F. Numerical Results*

We now present the results from the simulation runs. First, we show the evaluation of the metrics from the viewpoint of the VNF placement problem. The differences between the pure GA and GA with EvoVNFP (denoted as *GA+EvoVNFP*) are that the individuals are reinitialized at every traffic change (arrival/departure of requests) and there are no periodic goal changes. The results in the figures below are obtained as average values of 100 simulation runs.

First, we show the results of the success probability for calculating placements in Fig. 7. The x-axis shows the request arrival rate, the y-axis the success probability, and for each plot we also show the 95% confidence intervals. In both graphs, the results of GA+EvoVNFP are consistently higher than GA at all arrival rates. Furthermore, Fig. 7 shows that when the arrival rate increases, GA+EvoVNFP maintains high values while those of GA significantly decrease. This implies that GA+EvoVNFP can follow and adapt to heavy traffic dynamics, i.e., rapid arrivals and departures of the requests.

Next, we show the results of the performance of the calculated placements in Fig. 8. The results are represented by CDFs where the x-axis represents various metrics and the y-axis shows the cumulative probability. Note that GA+EvoVNFP has a disadvantage in this comparison because we only show the results of feasible solutions. Considering the results in Fig. 7, GA+EvoVNFP can generate solutions of difficult situations, i.e., situations where the methods have to place many chains. To generate the placement for these situations, GA+EvoVNFP needs to utilize many cores for placing all chains. Moreover, the side effect is that the delay becomes larger in such dense placements. In spite of this disadvantage, the graphs in Figs. 8(a) and 8(b) indicate that the results of GA+EvoVNFP are better regarding the number of cores and delays than those of GA at the different arrival rates. This is because the mechanism of fluctuating goals in EvoVNFP that helps reducing the calculation time works well. In summary, GA+EvoVNFP can generate placements which have better performance while following the dynamics of arrivals.

We now show the results of the cost for reconfiguration in Figs. 8(c) and 8(d) that are also represented by CDFs in the same way as before. For both of the number of migrations and resizings, we can see that the results of GA+EvoVNFP are better than those of GA at all arrival rates. The reason for this is that a placement for an epoch is very different from that of its subsequent epoch due to the reinitialization of populations in GA. These results show that placements generated by GA+EvoVNFP can adapt to the dynamic arrivals and departures of chains with less placement changes.

Next, we discuss the results of the number of used PMs and VMs shown as CDFs in Figs. 9(a) and 9(b), respectively. In Fig. 9(a), we can see that GA+EvoVNFP uses a slightly smaller number of PMs than GA, which means that, in the individuals generated by GA+EvoVNFP, the PMs are not widely distributed in the physical machines. Furthermore, Fig. 9(b) shows that GA+EvoVNFP uses almost the same number of VMs as GA. Combined with the result shown in Fig. 9(a), it means that each PM in the placements generated by GA+EvoVNFP uses a higher number of VMs than GA. Generally, the more distributed the PMs, VMs, and components are in the physical network, the more easy it is to adapt to the new situations because we can use extra unused resources to change the placements. However, the results show that distributing the components is not the strategy that GA+EvoVNFP uses to adapt to dynamic changes.

The results of Genetic Variation are shown in Fig. 9(c). The x-axis and y-axis of the graph in Fig. 9(c) represent the values of the Genetic Variation and the cumulative probability, respectively. From Fig. 9(c), we can see that GA+EvoVNFP has smaller values than GA. This is because GA initializes all individuals in the population when the requests change. This result implies that the diversity of the individuals in the population does not lead to the high adaptability of EvoVNFP.

Finally, we show the results of entropy $H(X_i|T)$ and mutual information $I(X_i;T)$ in Fig. 9(d) and Fig. 9(e), respectively. The results in Fig. 9(d) show that $H(X_i|T)$ of GA+EvoVNFP is the smallest of both values. Moreover, the shapes of CDF plots of EvoVNFP in Fig. 9(e) show that most of the results of $I(X_i;T)$ are low values but a few of the results of GA+EvoVNFP are high values. This means that GA+EvoVNFP has a few positions whose $H(X_i|T)$ are low and $I(X_i;T)$ are high, i.e., they are triggers. This is the mechanism which enhances the adaptability of EvoVNFP so that it can generate placements which have a few triggers and can easily adapt to the dynamics by only changing the values of the triggers. While GA also has a similar shape of $H(X_i|T)$ as those of GA+EvoVNFP, most of the values of $I(X_i;T)$ are high, which means that GA cannot generate small numbers of

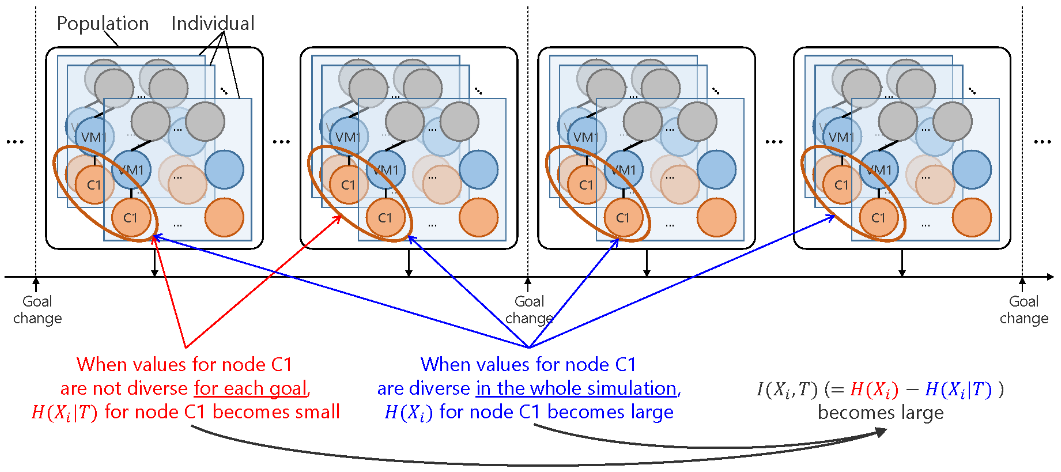

Population
Individual
VM1
C1
Goal change
Goal change
Goal change
When values for node C1 are not diverse for each goal, $H(X_i|T)$ for node C1 becomes small
When values for node C1 are diverse in the whole simulation, $H(X_i)$ for node C1 becomes large
$I(X_i,T)$ (= $H(X_i) - H(X_i|T)$ ) becomes large

Fig. 6. A schematic explanation of detection of triggers with entropy, conditional entropy, and mutual information

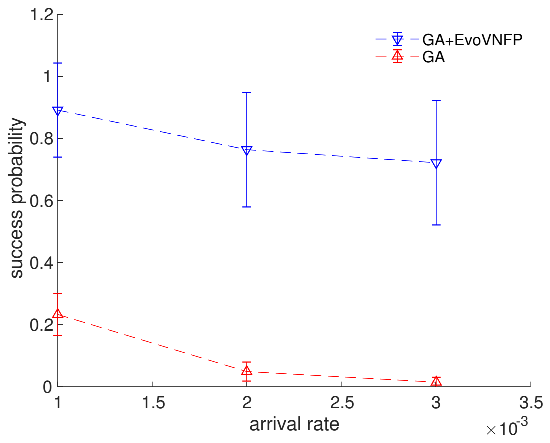


Fig. 7. Evaluation results of success probability over arrival rate

triggers in appropriate positions.

## G. Discussion

As a result of the evaluation shown above, we found that EvoVNFP has a mechanism for generating a small number of triggers among its individuals. Surprisingly, EvoVNFP does not have any other mechanisms which can improve the performance of the dynamic VNF placement problem, for example by widely distributing the components to the physical network or diversifying the individuals in the population. It seems that GA+EvoVNFP has a few kinds of individuals which have a small number of triggers within the population. When arrivals or departures of requests occur, GA+EvoVNFP can quickly adapt to the new situation because the population of GA+EvoVNFP has a lot of individuals with triggers and therefore the probability that mutations and crossovers change the values of triggers is high while keeping the number of triggers in each individual small.

## VI. Comparative Evaluation of EvoVNFP Performance

In this section, we show the results of evaluation of performance of EvoVNFP by combining it with two state-of-the-art iterative VNF placement methods utilizing GA and tabu search [16], [17]. Their respective algorithms are shown in Sec. III-C. We selected these two methods because their mechanisms significantly differ: the GA is based on selection and the tabu search is based on neighbor searches. In general, the GA is better than the tabu search when the solution spaces of problems are larger and more complex and it is therefore difficult to find optimal or suboptimal solutions with neighbor searches. We used the source code of the simulation program provided by the authors of ref. [16] for implementation in order to have a fair comparison [32]. For this evaluation, we had to slightly simplify the VNF placement problem compared to Sec. V. In the remainder of this section, we explain the modified models, as well as the simulation settings and evaluation metrics before we discuss the results.

### A. System Model

Each PM has an available capacity and each physical link has an available bandwidth in this model. Similarly, we set the required capacity of each VNF in the chains and the required bandwidth of each virtual link in the chains. We define the VNF placement problem as how to place the VNFs and virtual links on PMs and physical links, respectively, so that the sum of the required capacity or bandwidth on each PMs or each physical link does not exceed its available capacity.

In this model, we assume that there is a fixed number of chains in the physical network and the required capacity and bandwidth of the chains may change over time. When the required capacity or bandwidth changes, it may become necessary to reassign the VNFs and virtual links to PMs and physical links, respectively, in order to assure sufficient performance of the chains.

### B. Simulation Settings

The physical network is in this study a fat tree consisting of 64 PMs as shown in Fig. 10. The available capacity of each PM is 900 units and the available bandwidth on each physical link is 3000 units. There are 88 chains in the system, where each chain contains 2–7 VNFs. The choice of these parameters was

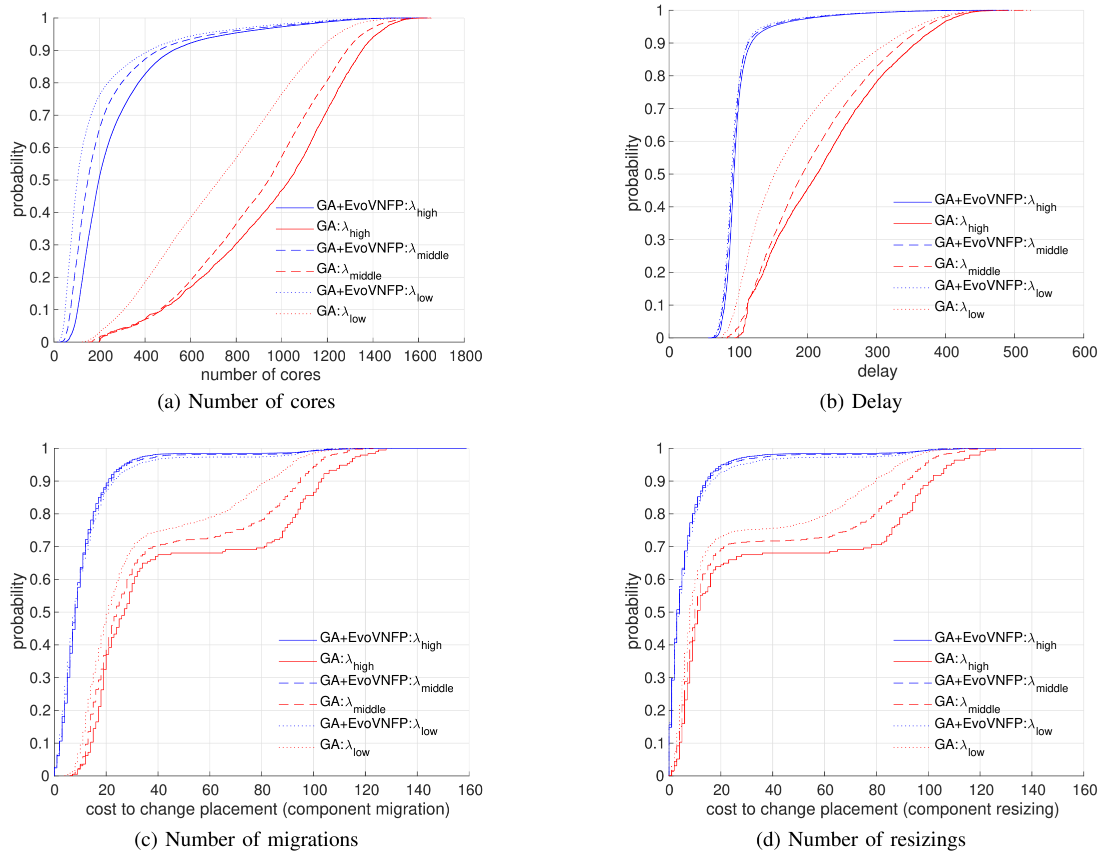


Fig. 8. CDFs of performance metrics of GA and GA+EvoVNFP for the VNF placement problem. Three different arrival rates of $\lambda_{low}$ (1 request per 1000 generations), $\lambda_{middle}$ (1 request per 500 generations), and $\lambda_{high}$ (1 request per 333 generations) are considered for both methods.

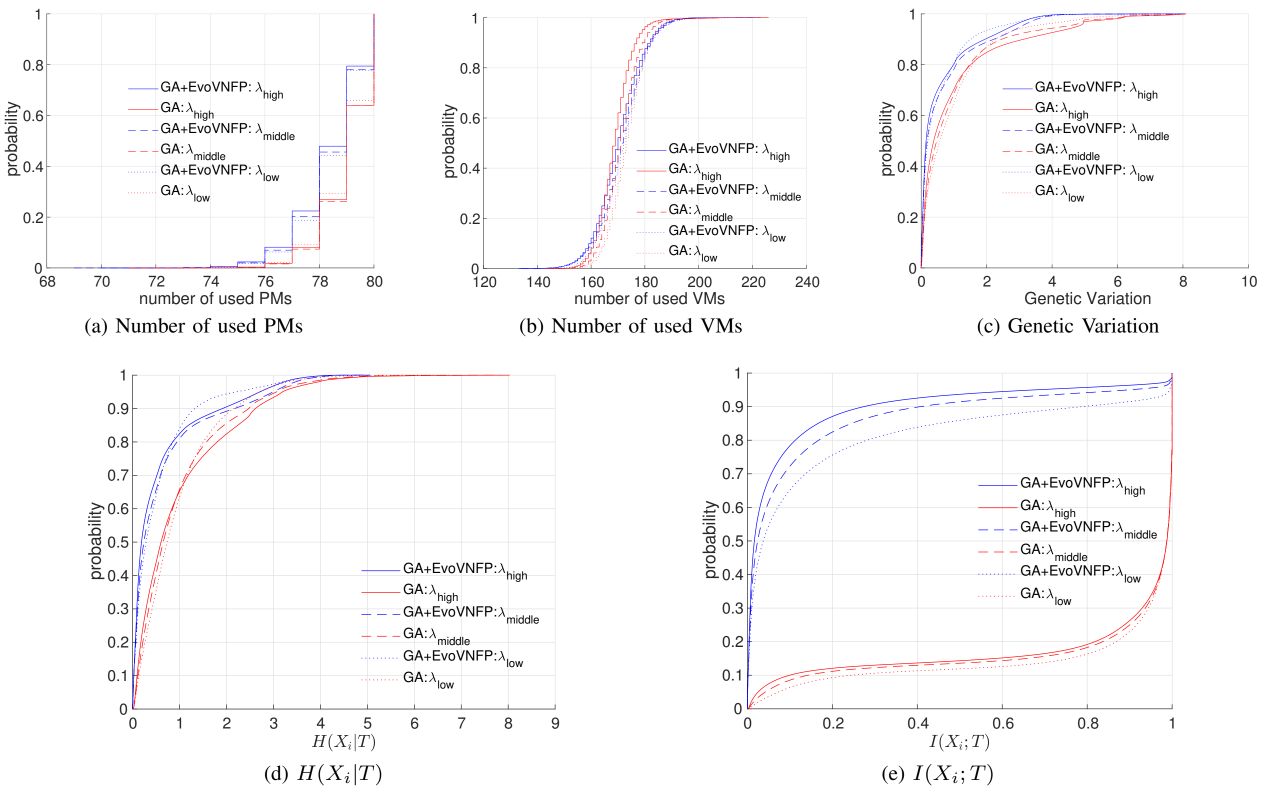


Fig. 9. CDFs of evaluation metrics for characterizing the internal structure of individuals. Similar to Fig. 8, three different arrival rates of $\lambda_{low}$ (1 request per 1000 generations), $\lambda_{middle}$ (1 request per 500 generations), and $\lambda_{high}$ (1 request per 333 generations) are considered for both methods.

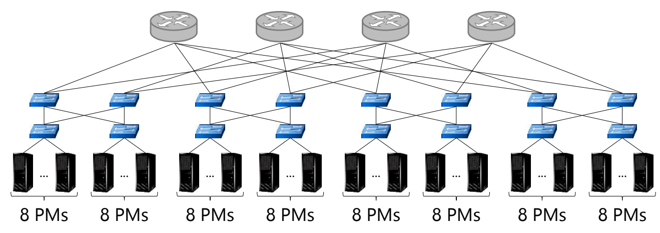


Fig. 10. Fat tree physical network for comparative evaluation

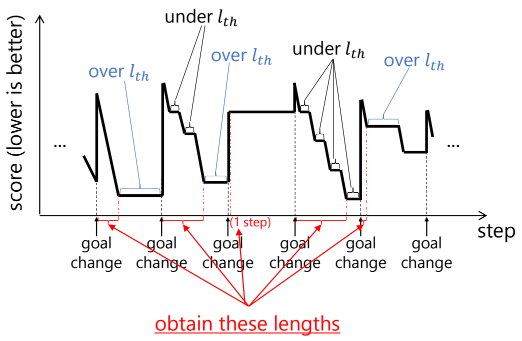


Fig. 11. Illustration of how to calculate number of steps until convergence

inspired by the settings in [16]. The system receives an input with changes of several chains at every fixed time step. An input includes two kinds of information: the variation amount of the required bandwidth for the selected chains and whether the number of VNFs in each chain has to be scaled up/down or not. The required capacity of each VNF is initially set to 100 units and increased to 200 units or decreased to 50 units. The required bandwidth of each link is initially selected from the range [1, 30] and the amount of the variation in scaling up/down is selected from [1, 30]. We consider six different values regarding the number of modifications to the chains at a time, i.e., 5, 10, 15, 20, 25, and 30 changes.

The evaluation function of these reference methods is based on the one in [16], [33], [34] as follows:

$$F = \frac{T_s}{M} + \frac{U_l}{L} + \frac{(L - T_l)}{L} + \frac{C_s}{M} + \frac{C_l}{L}, \tag{9}$$

where $M$ and $L$ are the number of PMs and links in the physical network, respectively. $T_s$ and $T_l$ are the number of used PMs and used links, $U_l$ is the bandwidth usage, and $C_s$ and $C_l$ represent how many VNFs and chains are on different PMs and links in the current placement compared to the previous placement, respectively.

Equation (9) contains five terms of which the first three are similar to the fitness function in Eq. (5), while the last two terms represent the differences between successive placements. We added these terms to Eq. (9) in order to include the effects of adaptability on the calculation of the score $F$.

*1) GA Settings:* The number of individuals in the GA population is 50 and only the best individual is passed as elite to the next step without mutation. At each step, one individual is selected from the non-elite population and mutated. The stopping criterion is reached after one epoch of 500 steps.

*2) Tabu Search Settings:* The size of the tabu list is 7, which is known to be the best value for general applications [35]. The tabu list stores migrations of VNFs in a chain to new PMs. The neighborhood of a placement is composed of all placements that are created from the original placement by randomly selecting a chain and migrating all VNFs in the chain to all other PMs. The stopping criterion is reached after 500 steps.

The settings of EvoVNFP are as follows. The length of the periods $T_p$ is 100. The number of chains removed in the original goals when we relax them is 1. We investigated also other values, but found that these are the most suitable.

## C. Evaluation Metrics

Finally, we use the following metrics in this evaluation:

*1) Number of steps to converge:* Figure 11 shows how we calculated this metric. We calculated it for each epoch. First, we set a value $l_{th}$ to judge whether the solution converge or not. When the score does not change for over $l_{th}$ steps, we judge the solution is converged. Then we obtain the number of steps between the beginning of the epochs and the timing where the converged solutions are generated. The fourth epoch in Fig. 11 is a special case. The score does not stays the same for over $l_{th}$ steps. In this case, we obtain the length between the beginning of the epochs and the timing where the final solutions in the epochs are generated.

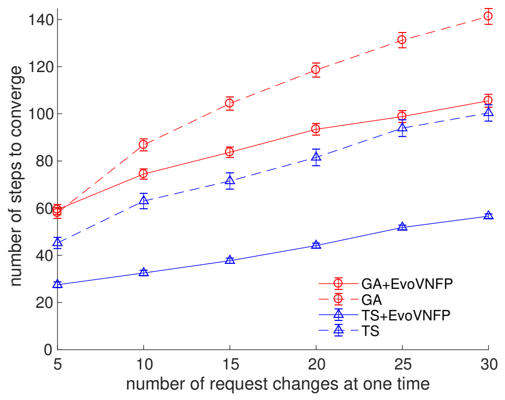
*2) Mean score of converged solutions:* The mean score of all converged solutions in all epochs. We judge the convergence according to the same way in the case of the number of steps to converge.

## D. Numerical Results

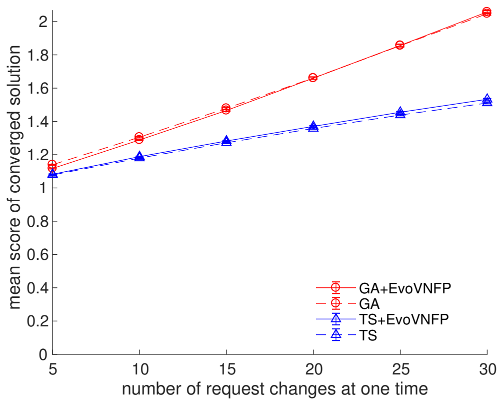
We discuss the results of the evaluation of the performance of EvoVNFP here. These results are obtained from 300 simulation runs. The x-axes represent the number of request changes input to the system at one time and the y-axes represent values of the metrics and for each plot we also show the 95% confidence intervals. We set $l_{th}$ to 100 steps.

First, we show the number of steps to converge in Fig. 12(a). In most cases, GA+EvoVNFP and TS+EvoVNFP are better than their versions without EvoVNFP. GA+EvoVNFP improves GA by up to 25% and TS+EvoVNFP improves TS by up to 48%. This means that the mechanism of EvoVNFP which periodically changes the goals for speedup works effectively. When we compare the methods based on the GA and the ones based on the tabu search, tabu search are better due to the fewer number of steps.

Second, we show the results of the mean score of converged solutions in Fig. 12(b). The plots for GA+EvoVNFP and GA are almost the same, same as TS+EvoVNFP and TS. We explain the reason in the following. When we see these results in more detail, i.e., checking the results of each term in the evaluation function, we can see that GA+EvoVNFP and TS+EvoVNFP use more PMs and links than GA and TS, respectively. The methods with EvoVNFP usually use more resources in order to increase adaptability. According to Eq. (9), the more links are used, the better the score becomes. Furthermore, using many links leads to smaller bandwidth usage. That is why the results become almost the same. When



(a) Number of steps to converge



(b) Scores of converged solutions

Fig. 12. Results of evaluation of performance of EvoVNFP

we compare the methods based on GA and the ones based on tabu search, those based on tabu search are better on the whole. Considering that they are also better in terms of the number of steps to converge, it seems that the shape of the solution space is more suitable for tabu search than for GA.

In terms of the runtime, the methods with EvoVNFP take longer to find a solution than those without. This is expected because EvoVNFP requires a small computational overhead. When we compare the methods based on GA to tabu search, GA performs better on the whole. This is because the calculation of the neighborhood in tabu search takes a lot of time. From the results above, tabu search can obtain better solutions but at the cost of a higher computation time compared to GA. Therefore, it would be better to select the most appropriate method according to the situation.

## VII. Conclusion

In this paper, we evaluated the mechanism and the performance of EvoVNFP. First, we investigated the mechanisms of EvoVNFP that enhance the adaptability of underlying methods by analyzing the structure of the placements which the GA combined with EvoVNFP generates. Second, we evaluated how much EvoVNFP improves underlying methods when combining it with two state-of-the-art VNF placement methods which are based on GA and tabu search, respectively. As a result of our evaluation, it is found that the placements generated by the GA with EvoVNFP include triggers, which assist in the adaptation toward new goals. Furthermore, it is also found that, regarding both methods, EvoVNFP can reduce the number of steps for calculation without decreasing the goodness of the placements. Our future work includes incorporating other effective strategies to EvoVNFP in order to further enhance its adaptability while maintaining the mechanism of triggers.

## Acknowledgment

This research was supported by "Program for Leading Graduate Schools" of the Ministry of Education, Culture, Sports, Science and Technology, Japan, and "Research and Development of Innovative Network Technologies to Create the Future", the Commissioned Research of the National Institute of Information and Communications Technology (NICT), Japan.

## References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surv. Tutor.*, vol. 18(1), pp. 236–262, Jan. 2016.
[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.
[3] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: A survey," *IEEE Commun. Mag.*, pp. 24–31, Nov. 2013.
[4] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu, "Research directions in network service chaining," in *Proc. IEEE SDN4FNS*, Nov. 2013.
[5] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep packet inspection as a service," in *Proc. ACM CoNEXT*, Dec. 2014.
[6] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. IEEE CNMS*, Nov. 2014, pp. 418–423.
[7] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service-chaining strategies using virtual network functions in operator networks," *Comput Netw.*, vol. 133, pp. 1–16, Mar. 2018.
[8] D. Cho, J. Taheri, A. Y. Zomaya, and L. Wang, "Virtual network function placement: Towards minimizing network latency and lead time," in *Proc. IEEE CloudCom*, Dec. 2017, pp. 90–97.
[9] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE CloudNet*, Oct. 2015, pp. 171–177.
[10] J. Cao, Y. Zhang, W. An, X. Chen, Y. Han, and J. Sun, "VNF placement in hybrid NFV environment: Modeling and genetic algorithms," in *Proc. IEEE ICPADS*, Dec. 2016, pp. 769–777.
[11] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1346–1354.
[12] "GS NFV-SWA 001—v1.1.1—network functions virtualisation (NFV); virtual network functions architecture," ETSI, Dec. 2014.
[13] B. Zhang, J. Hwang, and T. Wood, "Toward online virtual network function placement in software defined networks," in *Proc. IEEE/ACM IWQoS*, Jun. 2016, pp. 1–6.
[14] Y. Liu, H. Zhang, H. Guan, and Y. Wang, "A method for adaptive resource adjustment of dynamic service function chain," *IEEE Access*, Nov. 2018.
[15] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[16] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 343–356, Jun. 2017.

[17] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. IEEE NetSoft*, Apr. 2015, pp. 1–9.

[18] N. Kashtan and U. Alon, "Spontaneous evolution of modularity and network motifs," *Proc. Natl. Acad. Sci. USA*, vol. 102, no. 39, pp. 13 773–13 778, Sep. 2005.

[19] N. Kashtan, E. Noor, and U. Alon, "Varying environments can speed up evolution," *Proc. Natl. Acad. Sci. USA*, vol. 104, no. 34, pp. 13 711–13 716, Aug. 2007.

[20] M. Otokura, K. Leibnitz, Y. Koizumi, D. Kominami, T. Shimokawa, and M. Murata, "Application of evolutionary mechanism to dynamic virtual network function placemen," in *Proc. IEEE CoolSDN*, Nov. 2016.

[21] ——, "Impact of fluctuating goals on adaptability of evolvable VNF placement method," in *Proc. ASON*, Nov. 2016, pp. 304–310.

[22] C. Dovrolis and J. T. Streelman, "Evolvable network architectures: What can we learn from biology?" *Comput. Commun. Rev.*, vol. 40, no. 2, pp. 72–77, Apr. 2010.

[23] M. Parter, N. Kashtan, and U. Alon, "Facilitated variation: How evolution learns from past environments to generalize to new environments," *PLoS Comput. Biol.*, vol. 4, no. 11, p. e1000206, Nov. 2008.

[24] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surv. Tutor.*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[25] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," in *Proc. IFIP Networking*, May 2018.

[26] ——, "Charting the complexity landscape of virtual network embeddings," in *Proc. IFIP Networking*, May 2018.

[27] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Natl. Acad. Sci. USA*, vol. 103, no. 23, pp. 8577–8582, 2006.

[28] P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, "Structure and dynamics of core/periphery networks," *J. Complex Netw.*, vol. 1, no. 2, pp. 93–123, 2013.

[29] "European Topology map (pdf) Dec 2018," https://www.geant.org/Resources/Documents/GEANT_Topology_Map_December_2018.pdf.

[30] "T series Core Routers: T320, T640, T1600, TX Matrix, and TX Matrix Plus," https://lafibre.info/images/datacenter/200908_juniper_routeur_t1600.pdf.

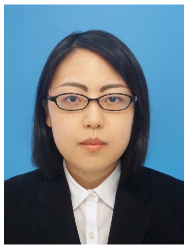
[31] C. Wang, J. Llorca, A. M. Tulino, and T. Javidi, "Dynamic cloud network control under reconfiguration delay and cost," *CoRR*, vol. abs/1802.06581, 2018.

[32] "GP based resource allocation for VNFs, starts with an DFS initial solution, continue with a simple scaling method," https://github.com/windyswsw/GPwithDFSandScaling.

[33] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *Proc. IFIP/IEEE IM*, May 2015, pp. 89–97.

[34] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Experimental results on the use of genetic algorithms for scaling virtualized network functions," in *Proc. IEEE NFV-SDN*, Nov. 2015, pp. 47–53.

[35] F. Glover, "Tabu search: A tutorial," *Interfaces*, 1990.



**Mari Otokura** is currently a third year doctor course student at the Graduate School of Information Science and Technology, Osaka University. Her research interests include modeling and performance analysis of communication networking, especially the application of biological evolvability to network virtualization.

**Kenji Leibnitz** received his M.Sc. and Ph.D. degrees in information science from the University of Würzburg, Germany. After joining the reserach group of Prof. Murata at Osaka University in May 2004, he joined the National Institute of Information and Communications Technology (NICT) in 2010. Since April 2013 he is with the Center of Information and Neural Networks (CiNet) of NICT and Osaka University. His research interests include the modeling and performance analysis of communication networks, especially biologically and brain inspired mechanisms for self-organization in future networks.

**Yuki Koizumi** is an associate professor of Graduate School of Information Science and Technology, Osaka University, Japan. Prior to that, he worked as an assistant professor at Osaka University. He received his master and Ph.D. degrees in information science from Osaka University in 2006 and 2009, respectively. His research interests include information centric networking and mobile networking.

**Daichi Kominami** received his M.E. and D.E. degrees from Osaka University, Japan, in 2010 and 2013. He is currently an Assistant Professor at the Graduate School of Economics, Osaka University, Japan. His research interests include distributed control in communication networks.

**Tetsuya Shimokawa** received his M.E. and Ph.D. degrees in engineering from Osaka University in Japan, after which he was a Research Assistant and Specially Appointed Associate Professor at Osaka University. Since April 2010 he is a Senior Researcher at NICT and from April 2013 he is with the Center of Information and Neural Networks (CiNet) of NICT and Osaka University. His interests include the mathematical modeling of the role of stochastic information processing in cognitive functions of the brain.

**Masayuki Murata** received the M.E. and D.E. degrees from Osaka University, Japan, in 1984 and 1988, respectively. In April 1984, he joined Tokyo Research Laboratory, IBM Japan, as a Researcher. He returned to Osaka University and was from 1987 to 1989 an Assistant Professor at the Computation Center, and from 1989 to 1999 at the Department of Information and Computer Sciences. In April 1999, he became a Professor of the Cybermedia Center and he is since April 2004 with the Graduate School of Information Science and Technology. He has more than five hundred papers of international and domestic journals and conferences. His research interests include computer communication network architecture, performance modeling and evaluation.