Application of Evolutionary Mechanism to Dynamic Virtual Network Function Placement

Mari Otokura*, Kenji Leibnitz†, Yuki Koizumi*, Daichi Kominami‡, Tetsuya Shimokawa† and Masayuki Murata*
*Graduate School of Information Science and Technology, Osaka University, Japan
E-mail: {m-otokura,ykoizumi,murata}@ist.osaka-u.ac.jp
†Center for Information and Neural Networks, NICT and Osaka University, Japan
E-mail: {leibnitz,shimokawa}@nict.go.jp
‡Graduate School of Economics, Osaka University, Japan
E-mail: d-kominami@econ.osaka-u.ac.jp

***Abstract*—Recently, communication network services have become increasingly diverse and dynamic. *Network Function Virtualization* (NFV) is an effective technique to deal with these dynamic situations. Most related work on the VNF placement problem does not consider the dynamics of requests, but only static scenarios. The important goals of the dynamic VNF placement problem include accommodating new requests following the traffic dynamics and reducing the time to calculate solutions. To tackle this problem, we utilize the concept of *Modularly Varying Goals* (MVG), which is based on a *genetic algorithm* (GA) and generates solutions that can easily adapt to time-varying goals in short time. In this paper, we propose *Evolvable VNF Placement* (EvoVNFP) that applies the concept of MVG to the dynamic VNF placement problem to reduce the time to obtain solutions. Results from numerical evaluations show that our method is able to better follow the dynamics of VNF requests and also reduce time until adapting to successive objectives.**

I. Introduction

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

Network Function Virtualization (NFV) [1]–[3] is a suitable way to deal with this situation. The basic idea of NFV is to separate the network functions from their physical computational resources. These network functions are implemented in software running on *virtual machines* (VMs), called *Virtual Network Functions* (VNFs) and run on commodity physical computational resources. NFV enables to change the performance of network functions dynamically. Furthermore, NFV also provides the pipelining of virtual network functions referred as *Service Function Chaining* (SFC) [4], [5].

In the field of NFV, many researchers have tackled the VNF placement problem by deciding placements of VNFs according to a specific objective [6]–[11]. The VNF placement problem can be seen as an instance of the *facility location problem* [12] which is known to be an NP-hard problem. Furthermore, it is suggested in the ETSI NFV ISG specifications [13] that a VNF can be broken into smaller parts, leading to a higher complexity of the problem.

Moreover, we have to solve the VNF placement problem dynamically when considering SFC to deal with dynamic changes of requests from users [14]. Although there are many proposals on the VNF placement problem, almost all of them focus on a static situation. Whenever the requests for VNFs arrive or depart in the dynamic VNF placement problem, besides optimizing each placement according to the current objective, we have to also generate new placements following the dynamic changes of situations. Also, we should reduce the time to calculate new placements in order to provide continuous services. A simple way of dealing with this problem is by solving an optimization problem every arrival/departure of requests. However, this will be too computationally intensive, since already the static VNF placement problem itself is NP-hard.

To tackle this problem, we utilize knowledge from biological evolution: when organisms evolve towards varying environments, they spontaneously have the adaptability to the varying goals. Reference [15] proposes *Modularly Varying Goals* (MVG) based on a *genetic algorithm* (GA) to introduce the concept of biological evolution in varying environments. GA is an algorithm which evolves the population of individuals every generation so as to fit the goals (environments). They showed that genomes evolved under MVG form adaptable structures to varying environments and also MVG itself can speed up the evolution [16]. In our previous work, we proposed *Evolutionary Varying Goals* (EVG) [17], where we already showed that the basic MVG concept can be applied to a simple VNF placement problem. EVG can generate individuals in the GA which can adapt to varying goals with only a few genetic changes.

In this paper, we propose a method named *Evolvable VNF Placement* (EvoVNFP), which is an extended version of EVG and inspired by MVG, for the realistic dynamic VNF placement problem for SFC. In order to facilitate the convergence toward "good" solutions, EvoVNFP uses small periodic

objective changes during adaptation. Evaluations show that EvoVNFP can track the varying environments well and reduce the time to adapt to new objectives.

The rest of the paper is organized as follows. In Section II, we explain the modeling and formulation of NFV systems for SFC. We then introduce the proposed method EvoVNFP in Section III. In Section IV, we show the results of the evaluation by simulations and discuss them. We conclude this paper in Section V.

II. System Model

A. Modeling of NFV Systems for SFC

In this section, we will explain the model of the NFV system for SFC which is used in this research.

1) Schematic view of the system: Figure 1 shows a schematic view of the considered system. In this system, users request VNF chains from the system controller (Step 1) that breaks the VNFs into *components* (Comps) (Step 2) and then places them on the *physical machines (PMs)* in the physical network (Step 3). Traffic of a user enters the physical network through its ingress router, traverses components in its chain in a specified order, and leaves through its egress router. While the traffic passes through all components in the physical networks, two kinds of delays will occur: *propagation delays* and *queuing delays*. Propagation delay occurs when the traffic passes through links. Queuing delay occurs when the traffic is processed by network equipment, such as routers and VMs for which we will use analytical approximations. In this work, we regard only CPU as the computational resource. We assume that the VMs occupy CPU cores of the PMs and the components occupy CPU cores of the VMs. We can place multiple components on a VM and multiple VMs on a PM as long as there are sufficient remaining cores in the VM or PM. A schematic view of the occupation of CPU cores is shown in the lower right of Fig. 1.

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

3) Definition of request flows: We define *request flows* as flows of the traffic of the users who send *requests* to the controller. A request R_u of 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.

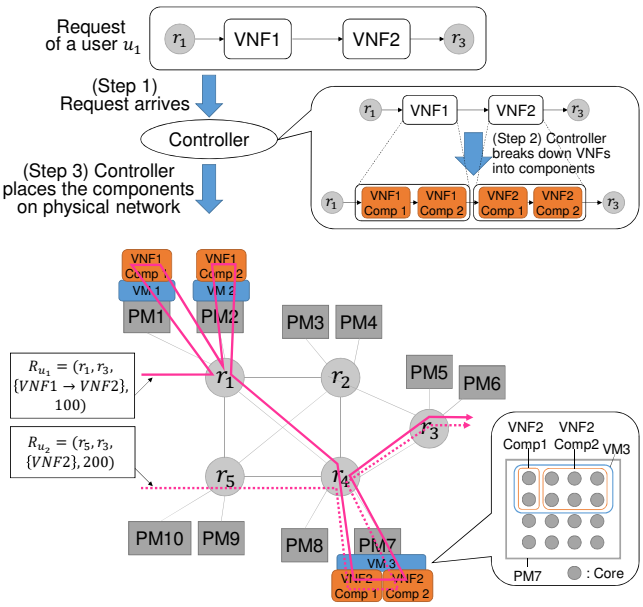


Fig. 1. A schematic view of how VNFs are broken down into components that are placed on virtual machines (VMs) at physical machines (PMs)

4) Calculating the delay of a placement: 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 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 λ_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 calculated first 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

The considered VNF placement problem is formulated as follows as an integer linear problem.

$$\text{minimize} \quad \hat{d} + W \cdot \sum_{i,k} m_{i,k} \tag{1}$$

$$\text{subject to} \quad T_a \cdot \frac{v_a}{S} \leq n_{k,j,a} \cdot C \qquad \forall k, j, a \tag{2}$$

$$\sum_k m_{i,k} \leq N_i \qquad \forall i \tag{3}$$

$$\sum_{j,a} n_{k,j,a} \leq m_{i,k} \qquad \forall k, i \tag{4}$$

$$\text{variables} \quad m_{i,k},\ n_{k,j,a},\ p^u_{(r,r')}$$

The term $m_{i,k}$ is the number of cores which the VM k occupies on PM i, W is a weighting coefficient for the sum of cores, and N_i is the number of cores of PM i. Constraint (2) means that the controller must determine the number of cores for each component such that it has enough performance to process all the request flows which pass through it. Constraint (3) means that the sum of the total number of cores for VMs on a PM must not exceed the total number of cores of the PM. Constraint (4) means that the sum of the number of cores for components on a VM must not exceed the number of cores of the VM.

III. Proposed Dynamic VNF Placement Method

In this section, we will propose a method named *Evolvable VNF Placement* (EvoVNFP) for the dynamic VNF placement problem. We assume a situation where requests arrive at the system, receive service for a certain time, and then depart from the system.

A. Overview of Proposed Method

EvoVNFP is based on EVG, which changes the objective of GA without re-initializing the population with every arrival/departure of requests and otherwise maintains the same objectives. EvoVNFP is different from EVG when there are more than two requests in the system; EvoVNFP changes its objective periodically as shown in "EvoVNFP simulation" in Fig. 2. We define the interval between the periodic changes of the objective as *period*. The initial state of the state transition is S_1 and it corresponds to when EvoVNFP considers all of the requests in the system. The other states correspond to when EvoVNFP considers requests excluding one randomly selected request. These state transitions are inspired by MVG in [16]. We include them to speed up the evolution of GA to obtain the placements for current objectives. The updates of real placements are done in state S_1 at two kinds of timings. One is when the first feasible solution for all of the requests at the time is obtained after the change of the objective, shown in Fig. 2 as the think black arrows between "Real placement" and "EvoVNFP simulation." Latency to accommodate a new request, shown in Fig. 2 as the dashed arrows, can occur because EvoVNFP can take some time to obtain a solution. The other update is done whenever a solution is found that improves the current solution by a certain percentage, shown as the thick gray arrows.

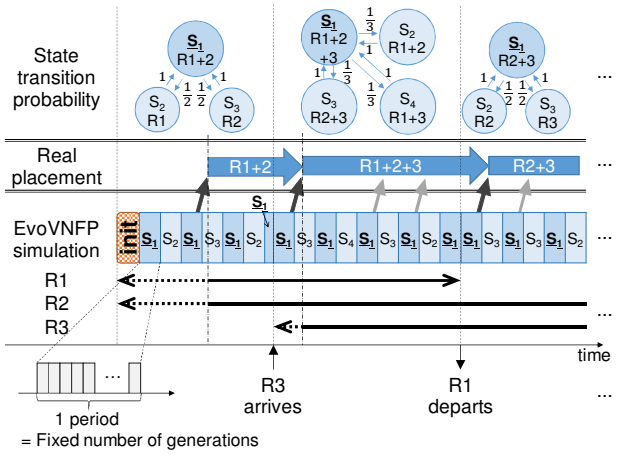


Fig. 2. A schematic view of EvoVNFP processing time-dependent arrivals and departures of requests. Ri denotes request i and Si denotes state i.

B. Design of Genetic Algorithm

In this section, we will explain the GA in the proposed method in detail. We will discuss how an individual is encoded and how the mutation and fitness function are defined.

1) Design of an individual: An individual in the GA is a fixed-length 3-layered graph and represents a placement, i.e., components assigned to VMs, which in turn are assigned to PMs (Fig. 3). The top layer represents the PMs, the middle layer represents the VMs, and the bottom layer represents the components. Each node corresponds to each PM, VM, and component in the placement. The component layer has enough nodes to accommodate all requests and these nodes and links connected to them can be unused, shown in Fig. 3 as white nodes and dashed links, respectively. They can connect to VM nodes to remember previous placements, but are ignored when the individual is converted to a placement. A node in the VM layer and a node in the PM layer are connected if the corresponding VM is placed on the corresponding PM. Similarly, a node in the component layer and a node in the VM layer are connected if the component is placed on the corresponding VM. If a node in the VM layer does not connect to the component layer, this node is ignored when the individual is converted to a placement. Furthermore, each node in the VM and component layers has the information about the number of cores which the corresponding VM or component occupies. The path of each request flow in the physical network is determined as the shortest path which connects all required components.

2) Fitness function: The fitness function evaluates how well an individual matches the objective. In our GA, it is a function

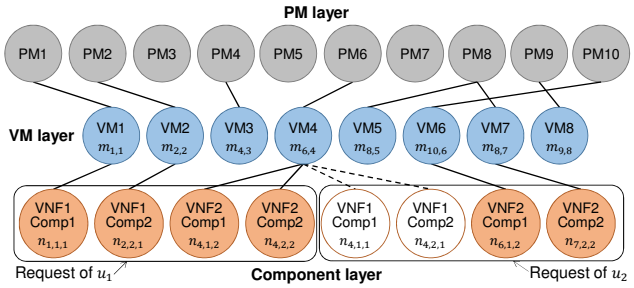


Fig. 3. An example of an individual representing a placement by a 3-layered graph.

shown in Eq. (5b).

$$F = \begin{cases} \left(\dfrac{\hat{d}}{d_{max}} + \dfrac{W(\sum_{i,k} m_{i,k})}{c_{max}} \right)^{-1} & \text{if the individual meets (2)–(4)} \quad (5a) \\ \alpha Z & \text{otherwise} \quad (5b) \end{cases}$$

where d_{max} is the maximum delay, i.e., the delay of a request flow where there is maximum possible number of components, each component performs with 80% utilization, and the hop length is the maximum hop length in the physical network plus three, c_{max} is the maximum possible number of cores, Z is the number of violations against the constraints, and α is a negative constant, e.g. $\alpha = -0.1$. Equation (5a) is used for individuals which can be converted to placements. The smaller the value of Eq. (1) is, the larger that of Eq. (5a) becomes. If the individual cannot be converted to a placement, Eq. (5b) will be used as the fitness function and yield a negative value. The higher the level of violation is, the lower Eq. (5b) becomes.

3) Mutation: In the mutation step of our GA, an individual is changed only slightly by mutations and without crossovers. Mutations are performed at a predefined mutation rate for all individuals except for the elites. We randomly select one of the four following kinds of operations for the mutation: i) changing a link between PM layer and VM layer randomly, ii) changing a link between VM layer and component layer randomly, iii) changing the number of cores of the node in VM layer randomly, and iv) changing the number of cores of the node in component layer randomly.

IV. Evaluation

In this section, we will first explain the settings of the simulations, the reference methods, and the evaluation metrics, and then present numerical results and their discussion.

A. Simulation Settings

We consider a physical network composed of five routers where each router is connected to two PMs as shown in Fig. 1. Each PM has sixteen CPU cores. The propagation delays of the physical link between two routers is 20 ms. The intervals of arrivals and sojourn time (in generations) of the requests in the system follow geometric distributions. There are policies to decide the order of VNFs in a chain [18]. Therefore, all requests in this evaluation use one of the four chains in the following: {VNF1}, {VNF1 → VNF2}, {VNF1 → VNF2 → VNF3}, {VNF1 → VNF2 → VNF3 → VNF4}. The parameters of the GAs are selected as follows: the number of individuals is 1000, the number of elites is 100, and the mutation rate is 0.8. The other parameters are chosen based on realistic values, such as currently available commercial machines or switches, and summarized in the following: requested transmission rate $b_u(t)$ = 200 Mbit/s of user u, processing capacity of a core $C = 3.0$ GHz, size of a packet $S = 1500$ bit, service rate $M_r = 3.0$ Gpacket/s of router r, and weighting coefficient $W = 1.0$.

B. Reference Methods

To evaluate the performance of our proposal, we will compare EvoVNFP to the following reference methods.

1) Conventional GA: Similar to EvoVNFP in Fig. 2, the objectives of conventional GA change every time the requests arrive or depart. The differences from EvoVNFP are that the individuals are re-initialized at every objective change (arrival/departure of requests), and there are no periodic objective changes. We use this as a reference method to clarify that forgetting past information is not a good strategy for the dynamic VNF placement problem.

2) Random Immigrant GA: In random immigrant GA [19], similar to EvoVNFP, the objectives change every time the requests arrive or depart and there are no re-initializations of the individuals. The differences from EvoVNFP are that it re-initializes some individuals at a predefined rate called *replacement rate* after every generation and there are no periodic objective changes. This method aims at generating diversity in the population to adapt to the varying environments.

C. Evaluation Metrics

We use the four evaluation metrics as follows. First, we evaluate *failure probability* over the cases where the methods cannot obtain any feasible solutions until the next change of the objectives. Here, a feasible solution refers to the individuals which have positive fitness, i.e., the individuals which can be converted to placements. Second, we evaluate the *number of generations until obtaining the first feasible solution.* Third, we evaluate the *number of reconfigurations,* which are needed to change placements between two successive requests. The reconfigurations include eight operations as shown in Table I. VM/component migrations mean that a VM or component, which had been placed on a PM, is moved to another PM. In this evaluation, we count inter-router and intra-router migrations separately because inter-router migrations take more time than intra-router ones [20]. VM/component size changes mean that the number of cores of a VM/component is increased or decreased. VM additions mean that the number of VMs is incremented in a placement and VM removals mean that the number is decremented. The weights are decided according to the time it takes to perform each operation [21] and the values we selected are shown in Table I. Finally, we evaluate two

TABLE I
WEIGHTS OF EIGHT OPERATIONS OF RECONFIGURATIONS

Kind of reconfiguration	Weight
Intra-router VM migration	30
Inter-router VM migration	120
VM size change	1
VM addition	60
VM removal	1
Intra-router component migration	30
Inter-router component migration	120
Component size change	1

measures of *performance for generated placements.* The first is the *number of cores in a placement* (system performance), which we evaluate through the number of physical cores used by VMs in each placement. The other is *delay of a placement* (user performance) described in Section II-A4.

D. Results and Discussions

In this subsection, we show the results of the simulations and their discussion. The results of EvoVNFP, conventional GA (Conv), and random immigrant GA (RandImm) are the averages of the results from 100 simulation runs. The replacement rate of RandImm is 0.3. Each run continues for 10000 generations. The evaluations use the results at 5000th–10000th generations to leave out results when the system is not in steady state. The length of the periods is 20 generations. Updates for improvement of real placements are performed if the newly computed solution of the methods is 10% better than the current real placement. These checks are done every 40 generations. There are 5 requests in the system at the beginning of the simulation. The maximum of the number of requests in the system is ten and we reject any requests that would exceed this number. In Fig. 4, the x-axis represents the generations and the y-axis represents the maximum fitness. In Figs. 5(a)–5(e), the x-axes represent the arrival rate and y-axes represent the average results of 100 simulations of each metric. The arrival rate of the requests used in the evaluations is 1/3000, 1/2500, 1/2000, 1/1500, 1/1000, and 1/500 (requests/generation). In order to obtain a constant offered load, the service rate per request of the system is one-fifth of the arrival rate. We evaluate only successful cases, i.e., when the system can find a solution within a period, in Figs. 5(b)–5(e). Note that, therefore, the results of Conv and RandImm tend to be better than EvoVNFP in Figs. 5(b)–5(e) because they solve only easy problems where the number of the requests is smaller.

1) An example of fitness transition: First, we show the results of the fitness of one simulation run where the arrival rate is 1/1000 in Fig. 4 to see differences in behavior among the methods. EvoVNFP fluctuates more than the other methods because of the periodic changes of the objectives. EvoVNFP and RandImm cannot generate solutions at first, but after about 1000 generations they generate feasible solutions and can maintain them because they don't re-initialize the population when the objectives changes. On the other hand, Conv cannot maintain the solutions because it re-initializes the population at every change of the objectives.

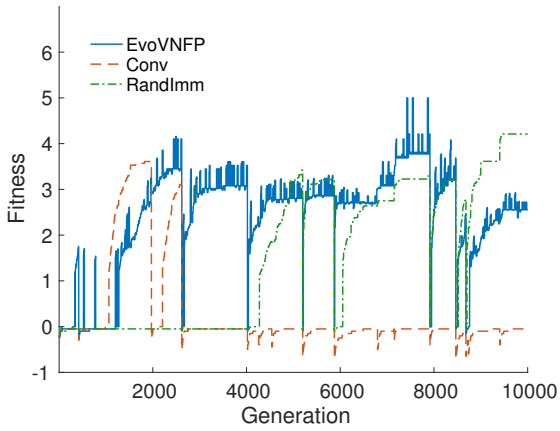
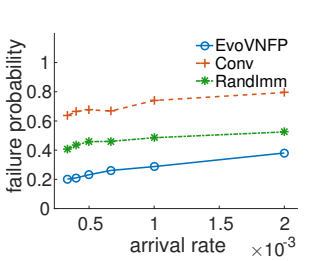


Fig. 4. An example of fitness transition

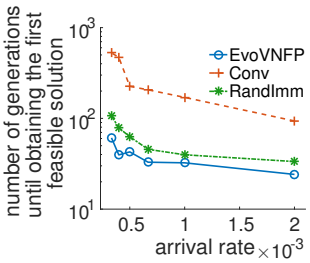
2) Evaluation of the time for successful adaptations and the failure probability: Second, we show the results of the number of generations until obtaining the first feasible solution for the new objectives in Figs. 5(a) and 5(b). For the higher arrival rates, the failure probabilities are higher in Fig. 5(a) because we have to find solutions in shorter time. Also, the number of generations is smaller in the higher arrival rates in Fig. 5(b) because all methods can solve easy problems where the number of requests is small. EvoVNFP shows the best performance among all three methods for all arrival rates in Figs. 5(a) and 5(b). Figure 5(b) shows the ability of EvoVNFP to reduce the time to calculate the first feasible solutions, which also leads to the lower failure probability in Fig. 5(a). The reason for this is that the periodic changes of EvoVNFP work effectively and prevent EvoVNFP from getting stuck in local solutions. Also, they facilitate the discovery of solutions because a solution for N requests can be easily created from a solution for $N-1$ requests, which are obtained from easier problems than the case of N requests.

3) Evaluation of the cost for reconfigurations: Next, we evaluate the total reconfiguration cost needed to adapt to succeeding objectives. It is measured using the last placements for preceding objectives and the first feasible placements for the new objectives. Figure 5(c) shows that RandImm has a slightly lower cost than EvoVNFP. It is assumed from this result that EvoVNFP generates placements where the components and VMs for each request are separated to different VMs and PMs, respectively, because of the periodic changes of the objectives. This makes the speed of adaptation faster because the resources for the requests do not interfere with each other. However, it also makes the difference between the placements larger. This is because recalling the separated resources causes sudden appearance of VMs and components, leading to many VM additions.

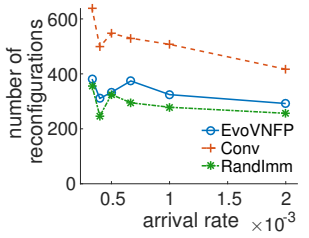
4) Evaluation of the performance of the placement: Finally, we evaluate the quality of the placement, i.e., the number of cores and delays, at the updates. EvoVNFP and RandImm are almost same and better than Conv in Fig. 5(d) and EvoVNFP



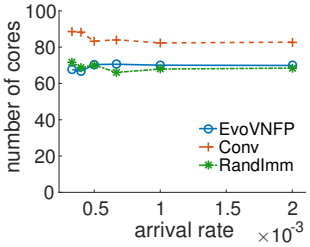
(a) Failure probability of request acceptance.



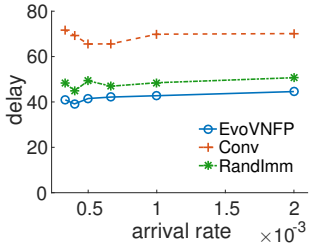
(b) Number of generations until obtaining the first feasible solution.



(c) Weighted sum of reconfigurations needed to adapt to succeeding objectives.



(d) Number of CPU cores of the placement at updates.



(e) Delay of the placement at updates.

Fig. 5. Evaluation results where x-axes represent arrival rate. The arrival rates used in the evaluations are 1/3000, 1/2500, 1/2000, 1/1500, 1/1000, and 1/500 (requests/generation) and marked in the figures. (b)–(e) are under conditions of success.

is the best in Fig. 5(e). These are because EvoVNFP can improve the individuals faster than RandImm and Conv, not only obtain the first feasible solutions faster. These results mean that EvoVNFP doesn't have overhead in terms of the system-side/user-side performance.

V. Conclusion and Future Work

In this research, we proposed a method called EvoVNFP for solving the dynamic VNF placement problem. EvoVNFP includes fine-grained changes of the objectives every period to facilitate fast adaptation. Also, when the requests arrive or depart, the objectives of EvoVNFP change without re-initialization of the population.

The results of the simulations show that EvoVNFP has smaller failure probability of finding feasible solutions, i.e. can follow the dynamics, and can reduce the adaptation time to successive objectives in comparison to conventional GA and random immigrant GA. Furthermore, EvoVNFP has virtually no overhead in terms of the performance.

Our future work includes further evaluations with different types of objective changes to measure adaptability of EvoVNFP and evaluate further metrics to show that EvoVNFP has good performance for solving the dynamic VNF placement problem.

Acknowledgment

This research was supported by "Program for Leading Graduate Schools" of the Ministry of Education, Culture, Sports, Science and Technology, 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 Communications Surveys & Tutorials*, 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 Communications Magazine*, 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 Communications Manazine*, 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 *Proceedings of SDN4FNS 2013*, Nov. 2013, pp. 1–7.

[5] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep Packet Inspection as a Service," in *Proceedings of CoNEXT 2014*, Dec. 2014, pp. 271–282.

[6] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions in NFV," in *Proceedings of CNMS 2015*, Nov. 2015.

[7] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near Optimal Placement of Virtual Network Functions," in *Proceedings of INFOCOM 2015*, Apr. 2015, pp. 1346–1354.

[8] M. Bouet, J. Leguay, and V. Conan, "Cost-based Placement of vDPI Functions in NFV Infrastructures," in *Proceedings of NetSoft 2015*, Apr. 2015, pp. 1–9.

[9] M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutaba, "Service Function Chaining Simplified," *eprint arXiv:1601.0075*, Jan. 2016.

[10] M. Bouet, J. Leguay, and V. Conan, "Cost-Based Placement of Virtualized Deep Packet Inspection Functions in SDN," in *Proceedings of MILCOM 2013*, Nov. 2013, pp. 992–997.

[11] H. Moens and F. De Turck, "VNF-P: A Model for Efficient Placement of Virtualized Network Functions," in *Proceedings of CNMS 2014*, Nov. 2014, pp. 418–423.

[12] C. Aikens, "Facility Location Models for Distribution Planning," *EJOR*, vol. 22, no. 3, pp. 263 – 279, Dec. 1985.

[13] *GS NFV-SWA 001 - V1.1.1 - Network Functions Virtualisation (NFV); Virtual Network Functions Architecture*. ETSI, Dec. 2014.

[14] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The Dynamic Placement of Virtual Network Functions," in *Proceedings of NOMS 2014*, May 2014, pp. 1–9.

[15] N. Kashtan and U. Alon, "Spontaneous Evolution of Modularity and Network Motifs," *PNAS*, vol. 102, no. 39, pp. 13 773–13 778, Sep. 2005.

[16] N. Kashtan, E. Noor, and U. Alon, "Varying Environments Can Speed Up Evolution," *PNAS*, vol. 104, no. 34, pp. 13 711–13 716, Aug. 2007.

[17] M. Otokura, K. Leibnitz, T. Shimokawa, and M. Murata, "Evolutionary Core-Periphery Structure and its Application to Network Function Virtualization," *IEICE Transactions on NOLTA*, vol. 7, no. 2, Apr. 2016.

[18] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment," in *Proceedings of HotNets-X*, Nov. 2011, pp. 21:1–21:6.

[19] J. Grefenstette, "Genetic Algorithms for Changing Environments," in *Proceedings of PPSN 1992*. Elsevier, Sep. 1992, pp. 137–144.

[20] J. Barrera, M. Ruiz, and L. Velasco, "Orchestrating Virtual Machine Migrations in Telecom Clouds," in *Proceedings of OFC 2015*, Mar. 2015, pp. 1–3.

[21] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A Cost-Aware Elasticity Provisioning System for the Cloud," in *Proceedings of ICDCS 2011*, Jun. 2011, pp. 559–570.