Impact of Fluctuating Goals on Adaptability of Evolvable VNF Placement Method

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*—Software Defined Network (SDN) and *Network Function Virtualization* (NFV) are effective techniques to deal with dynamically changing network environments. Furthermore, the combination of SDN and NFV permits telecommunication service providers to offer sequences of virtualized network functions to their users through *Service Function Chaining* (SFC). In the context of SFC, the *Virtual Network Function* (VNF) placement problem, i.e., determining where the virtual functions should be located in the network, needs to be solved dynamically whenever new function chains are requested. In our previous work, we proposed an evolutionary method for dynamic VNF placement problems named *Evolvable VNF Placement* (EvoVNFP). This current paper aims at evaluating EvoVNFP in greater detail to clarify the influence of the parameter settings on the performance of EvoVNFP. Results from computer simulations show that appropriate settings of sub-goal period lengths and number of mutations help improve the adaptability and convergence speed of EvoVNFP.**

***Keywords*-network function virtualization (NFV), software defined networking (SDN), evolvability, dynamic placement problem.**

I. Introduction

Recently, researchers have been increasingly conducting studies about *Software Defined Networking* (SDN) and *Network Function Virtualization* (NFV). SDN and NFV enable the network operator to reduce the capital expenditure (CAPEX) and operating expenditure (OPEX) and to flexibly control the traffic paths and the deployment of *virtual network functions* (VNFs). *Service Function Chaining* (SFC) [1], [2], in which traffic of each user is transmitted according to a sequence of requested VNFs, is realized by the combination of both technologies of SDN and NFV. The traffic paths traversing the user-requested functions are referred to as *chains*. Controlling SFC is equivalent to solving the *VNF placement problem* [3]–[9], where the allocation of VNFs to *virtual machines* (VMs) and VMs to physical networks, respectively, is decided according to the current network conditions. In terms of complexity, VNF placement problems are NP-hard as they belong to the category of *facility location problems* [10].

In the context of SFC, user requests for function chains are often not static, but changing over time. Therefore, the placements of VNFs need to be dynamically adapted according to the environmental changes. We define such type of problems as *dynamic VNF placement problems* [11]. For dynamic VNF placement problems, it is important to reduce the time needed to determine new placements, i.e., to increase the adaptability of the method to the system's dynamics, in order to follow the dynamic changes of the requests, as well as to improve the quality of the placements. A simple, but impractical, solution would be to repeatedly solve this optimization problem at every arrival or departure instant of a request. However, when the request dynamics change too frequently, this solution will fail to work well due to the long time required to solve the NP-hard VNF placement problem.

In our recent work [12], we proposed a faster method for dealing with dynamic VNF placement problems named *Evolvable VNF Placement* (EvoVNFP). EvoVNFP is inspired by biological evolution under varying environments [13], [14]. It is shown that biological organisms can evolve under such varying environments by forming modules that remain stable over time and by adapting to the environmental changes with only minimal changes among these modules. Inspired by this behavior, EvoVNFP toggles between two or more objectives by performing an *evolutionary algorithm* (EA) to make convergence of individuals toward their common modules faster [12]. We showed in [12] that EvoVNFP can outperform reference methods at a wide range of request arrival rates. However, we did not fully discuss in [12] the adaptability of EvoVNFP when offered loads and parameters are changed.

In this paper, our goal is therefore to evaluate EvoVNFP under different loads and parameter settings to discuss its adaptability in greater detail. Evaluations through simulations show that EvoVNFP performs better than the reference methods, conventional evolutionary algorithm and Random Immigrant Genetic Algorithm (GA) [15], in almost all of the considered load conditions and we identify specific parameter settings that can best enhance its adaptability.

The remainder of this paper is organized as follows. In Section II, we briefly explain the underlying system model, followed by the mathematical formulation of NFV systems

and the VNF placement problem. We then explain details of the EvoVNFP method in Section III followed by Section IV, where we present the results of the numerical evaluation and discuss them. Finally, we conclude this paper in Section V.

II. System Model and Definition of VNF Placement Problem

In the following, we will use the same NFV model as in [12] to briefly explain the underlying mathematical system model. First, we will provide an overview of the system itself which will be followed by the formulation of the VNF placement problem on this system.

A. Overview of the System

When a user requests a chain from the system's controller, the controller converts the chain to a format which can be physically placed on the servers; the VNFs are split to several functional parts which are denoted as *components*. Then, the controller tries to find an assignment of the components to VMs and further determines where these VMs should be located on *physical machines* (PMs) in the physical network, see Fig. 1. The components and VMs occupy CPU cores of the VMs and PMs which they are placed on, respectively. Multiple components can be placed on a VM (and VMs on a PM, respectively), as long as the VM or the PM has a sufficient number of remaining CPU cores. After the placement of components is completed, the service for the user who requested this chain will begin. Traffic of the user passes through the components consisting of the requested VNFs in a specific order. It enters the physical networks through an *ingress router* and exits from an *egress router*. In the considered system, two kinds of delay may occur, propagation delay on physical links and queuing delay in the VMs and components. We define the total delay of a chain as the sum of all propagation delays and queuing delays occurring on this chain.

B. Formulation of VNF Placement Problem

As shown in [12], we can formulate the VNF placement problem as an integer linear problem given below by Eqs. (1)–(4).

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

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

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

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

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

The term $\hat{d}$ in Eq. (1) is the average delay of all chains in the system, W is a weighting coefficient for the sum of cores, and $m_{i,k}$ is the number of cores that the VM k occupies on PM i. Constraint (2) represents that the controller must determine the number of cores $n_{k,j,a}$ for each component a of VNF j occupying VM k in such way that it has enough performance to process all the request flows which pass through it. C is the maximum number of instructions that a core can process per time unit (seconds), T_a is the processing amount needed by VMs with components of VNF a to process a packet, v_a is the traffic arrival rate at VNF a and S is the size of a packet. Constraint (3) means that the sum of the total number of cores for VMs on a PM must not exceed the total available number of cores N_i of the PM, and Constraint (4) indicates that the sum of the number of cores for components on a VM must not exceed the total number of cores of the VM.

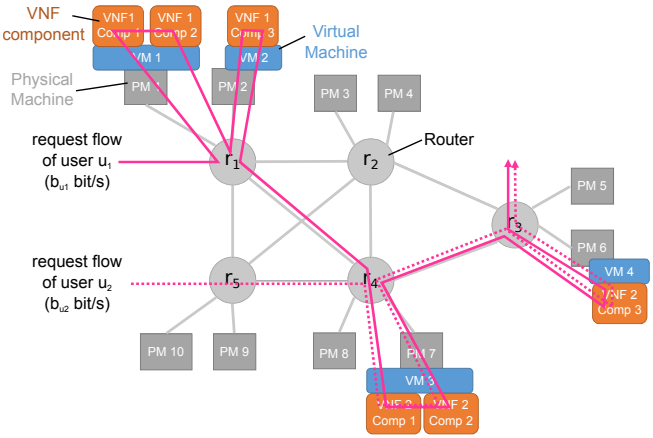

Figure 1. Example solution of the VNF component placement problem for two users. Each user's traffic flow passes through a chain of VMs hosting VNF components and located on physical machines (PMs).

Since it is an NP-hard problem, solution of this VNF placement problem is computationally intensive. For this reason, we will in the next section describe our proposed EvoVNFP method, which can heuristically solve this problem.

III. Dynamic VNF Placement Method: EvoVNFP

In this section, we will briefly explain the behavior of EvoVNFP as a solution method for the VNF placement problem given in Section II. For more details, the interested reader is referred to [12].

A. Overview of EvoVNFP

EvoVNFP is our proposed method for solving dynamic VNF placement problems. It uses a special *Evolutionary Algorithm* (EA), which is an algorithm imitating evolution in biological systems to generate solutions for an optimization problem. Similar to conventional EAs, our method is also composed of four major function steps: *initialization*, *evaluation*, *selection*, and *crossover/mutation*.

A single generation consists of performing the latter 3 of these 4 steps, and EAs are repeated over several generations

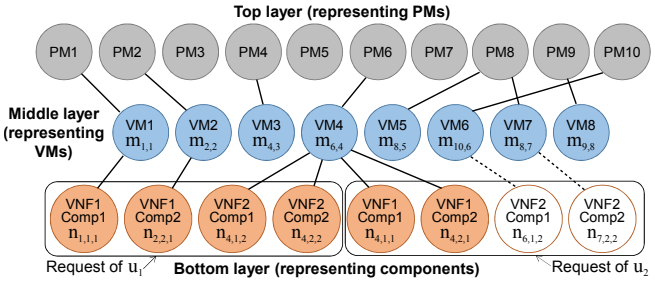

Figure 2. Example encoding of an individual in EvoVNFP. Three layers from top to bottom represent PMs, VMs, VNF components and links between nodes indicate how they are allocated to each other.

until a good solution in terms of the *fitness function* is obtained. The objective of the EA in this study is to generate as good as possible solutions for allocating the requests (chains) to the system's resources. While conventional EAs maintain the same objective function throughout operation and continuously evolve the individuals toward this single objective, our approach EvoVNFP toggles between multiple (sub-)objectives every fixed number of generations (*epoch*) when it evolves the individuals towards a main objective. The objective is continuously switched between the two following cases:

- generate a solution for the main objective which considers all of the current requests,
- generate a solution for the objective which considers all the current requests *except for one randomly selected request*.

Switching between these two types of objectives causes the formation of a modular structure corresponding to the specific subgoals, which in turn results in a faster adaptation of EA to the main objective. We will use the term *period* T_p for describing the interval (in generations) that the same goals remain active.

B. Design of Evolutionary Algorithm in EvoVNFP

We will now explain the key evolutionary concepts of EvoVNFP through the definition of individuals, the definition of the fitness function, and the mutation operator.

1) Individual: An individual of the EA is designed as a fixed-length 3-layered graph as shown in Fig. 2, representing a placement, i.e., the allocation of the VMs and components. The nodes in this graph represent physical and virtual network devices; the nodes in the top, middle, and bottom layer represent the PMs, VMs, and components, respectively. The nodes in the VM layer and component layer have information of the number of cores they occupy. Furthermore, the links between nodes represent their allocation; the links between the top and middle layer represent the allocation of the VMs to the PMs, and the links between the middle layer and the bottom layer represent the allocation of the components to the VMs. The white nodes in the bottom layer of Fig. 2 indicate that the VNF components corresponding to these nodes are currently not used in this placement. Such nodes and the dashed lines connecting them are ignored when the individual is converted to an actual placement.

2) Fitness Function: The fitness function F of the EA indicates how suitable an individual is for a specific goal. We define F in our case as

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

where d_{max} is the maximum delay, i.e., the delay of a request flow with the maximum possible number of components, each component is performing at 80% utilization, and the hop length is the maximum hop length in the physical network plus three because the paths of the chains can have loops and then we cannot define the maximum hop length as maximum chain length. Furthermore, 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., chosen as $\alpha = -0.1$.

This fitness function F will return a positive value if the individual can be converted to valid placements, otherwise F will be negative.

3) Mutation: The mutations in this EA consist of four operations as follows:

- selecting one link between the top and middle layer randomly and changing the PM node of this link randomly,
- selecting one link between the middle and bottom layer randomly and changing the VM node of this link randomly,
- changing the number of cores of a node in the middle layer,
- changing the number of cores of a node in the bottom layer.

In this work, we only focus on the above mentioned types of mutation operations and do not perform crossovers.

IV. Numerical Evaluations

In this section, we will first explain the simulation settings we assumed, the reference methods we will use for comparison, and the evaluation metrics. Then we will show the results and discuss them.

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 is equipped with sixteen CPU cores and the propagation delays over the physical link between two adjacent routers is 20 ms. Since we assume a discrete time system with time units in generations, the inter-arrival and

sojourn time of the requests in the system follow geometric distributions.

Usually, there are policies to decide the order of VNFs in a chain [16]. Therefore, all requests in this evaluation use one of the four chains in the following:

- $\{\text{VNF1}\}$,
- $\{\text{VNF1} \rightarrow \text{VNF2}\}$,
- $\{\text{VNF1} \rightarrow \text{VNF2} \rightarrow \text{VNF3}\}$, and
- $\{\text{VNF1} \rightarrow \text{VNF2} \rightarrow \text{VNF3} \rightarrow \text{VNF4}\}$.

The parameters of the evolutionary algorithms are selected as follows:

- number of individuals: 1000
- number of elites: 100
- mutation probability: 0.8

The other parameters are chosen based on realistic values and summarized below:

- value of $b_u(t)$: 200 Mbit/s
- processing capacity of a core: $C = 3.0$ GHz
- size of a packet: $S = 1500$ bit
- service rate of router r: $M_r = 3.0$ Gpacket/s
- weighting coefficient: $W = 1.0$

B. Reference Methods

To evaluate the performance of our proposal, we compare EvoVNFP to the following reference methods: conventional EA and Random Immigrant GA.

1) Conventional EA (Conv): Similar to EvoVNFP, the objectives of conventional EA change every time the requests arrive or depart at the system. However, 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 as in EvoVNFP. We use conventional EA as a reference method to clarify that entirely forgetting past information is not a good strategy for this dynamic VNF placement problem.

2) Random Immigrant GA (RandImm): In random immigrant GA [15], 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 RandImm re-initializes some individuals at a predefined rate called *replacement rate* after every generation and thus the objective changes are not periodic as in EvoVNFP. We include this method for comparison as it aims at generating diversity in the population to adapt to the varying environments.

C. Evaluation Metrics

We use three evaluation metrics to compare the performance among the considered methods: *failure probability*, *number of generations until obtaining the first feasible solution*, and *performance of generated placement*. Before we show the numerical results, we will first explain some further details on failure probability and the performance of generated placement in the following.

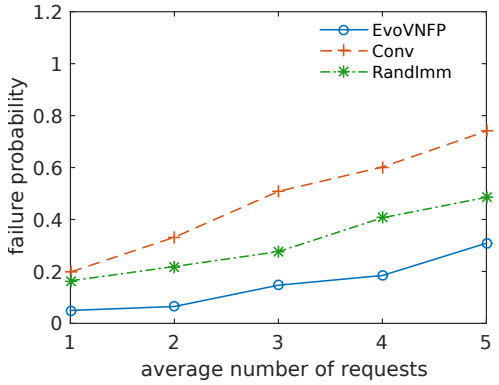

Figure 3. Failure probability of request acceptance, i.e., the ratio of the number of cases where the methods cannot obtain any feasible solutions until the next change of the objectives to the total number of changes of the objectives

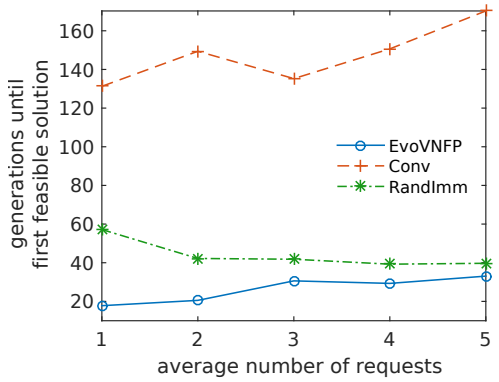

Figure 4. Number of generations until obtaining the first feasible solution

1) Failure probability: 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 having positive fitness, i.e., individuals which can be converted to valid placements.

2) Performance of generated placement: We evaluate two measures for computing the performance of generated placements. The first is the *number of cores in a placement*, which we evaluate through the number of physical cores used by VMs in each placement. This metric evaluates the placement from the system's viewpoint. On the other hand, we also include the user performance by considering the *delay of a placement*.

D. Results and Discussions

In this subsection, we present the results from the simulations and discuss them. We first show the comparison among EvoVNFP and its two reference methods. The results of EvoVNFP, conventional EA (Conv), and random immigrant GA (RandImm) are the averages of the individual results from at least 100 simulation runs. After these compar-

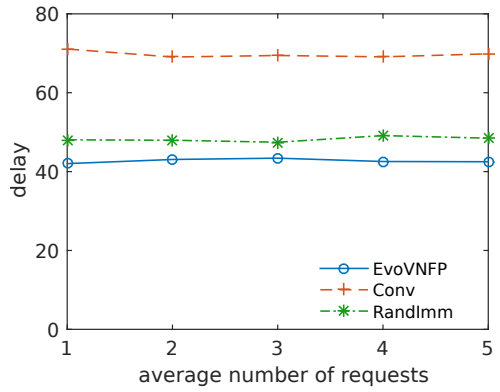

Figure 5. Delay of the placement at updates

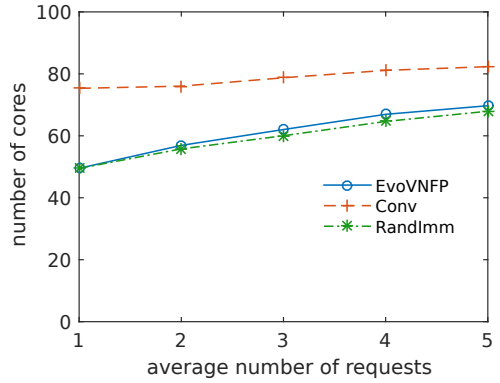

Figure 6. Number of CPU cores of the placement at updates

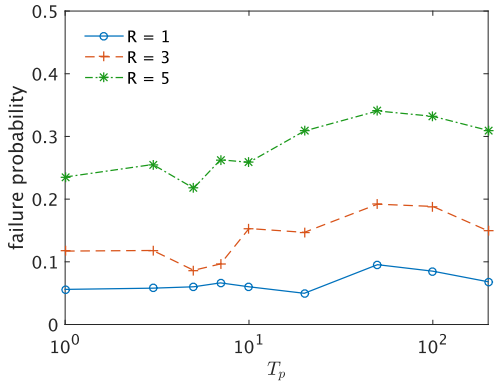

Figure 7. Failure probability of request acceptance, i.e., the ratio of the number of cases where the methods cannot obtain any feasible solutions until the next change of the objectives to the total number of changes of the objectives

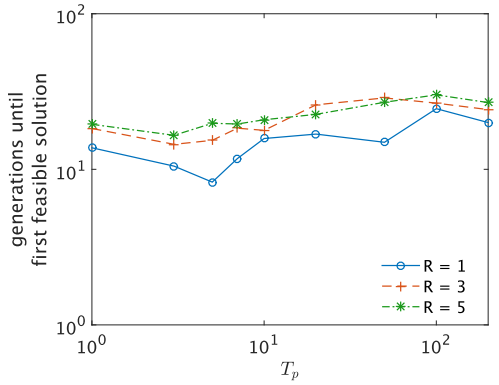

Figure 8. Number of generations until obtaining the first feasible solution

isons between methods, we focus on the performance of EvoVNFP. We consider different period lengths in order to discuss the impact of fluctuating goals on the adaptability. Finally, we show the results of EvoVNFP using different number of mutations in order to discuss their impact on adaptability.

We focused only on the above mentioned parameters and do not further discuss other parameters at this point, such as the number of individuals or the number of elites, because it is expected that these will not have a significant effect on the dynamics of the method. Moreover, we do not modify the mutation probabilities because our previous studies have shown that they don't have a significant impact on the results.

The replacement rate of RandImm is selected as 0.3. Each simulation run is performed over 10000 generations and we left out the first 5000 generations for the computation of our metrics as warm-up phase. Updates for improving the placements are performed only if the newly computed solution is at least 10% better than the current placement. Such checks are performed every 40 generations. Each simulation starts with initially 5 requests and any new requests exceeding the maximum number of ten are rejected.

Since the system is operating in discrete time units (generations), the simulation is performed over discrete time steps and we specify for each evaluation two parameters, T_A and T_B, denoting the average interarrival time and the average sojourn time of the system, respectively. Both values are obtained from geometrically distributed random numbers for which we give the mean durations $E[T_A]$ and $E[T_B]$.

1) Evaluation with varying system load: We first show the comparison of EvoVNFP and the reference methods under different load situations of the system in Figs. 3–6. The x-axes in these figures represent average loads of the system, i.e., the average number of requests in the system, and the y-axes represent the metrics explained in Section IV-C. In this study, we show the results when the average number of the chains ranges from 1 to 5, which are calculated using $E[T_A] = 5000$, 2500, 1666, 1250, and 1000, and $E[T_B] = 5000$ time steps. Furthermore, we use a fixed number of periods of $T_p = 20$ in these evaluations.

Figure 3 shows that the failure probability of EvoVNFP is lowest among the 3 considered methods. It means that EvoVNFP can follow the dynamics of the request arrivals/departures. Figure 4 shows the reason for the low failure probability, where EvoVNFP can evolve the indi-

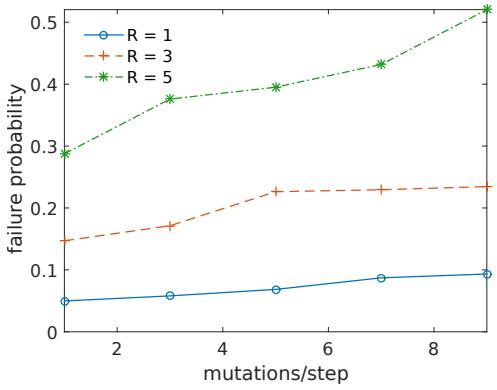

Figure 9. Failure probability of request acceptance, i.e., the ratio of the number of cases where the methods cannot obtain any feasible solutions until the next change of the objectives to the total number of changes of the objectives

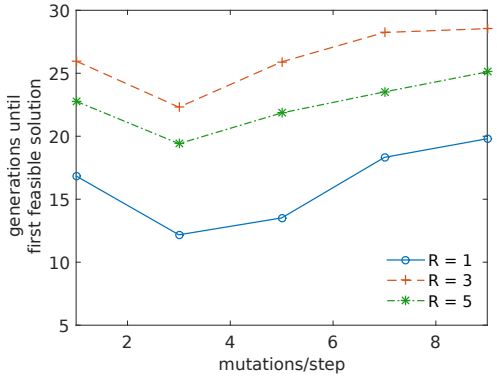

Figure 10. Number of generations until obtaining the first feasible solution

viduals towards the objective faster than the other methods due to the small fluctuation of the objective in the EA. EvoVNFP generates an individual for a simpler problem and then extends it to the main problem. We can also see that the failure probability increases when the average load increases. This is because the problem becomes more difficult when there are many chains are in the system.

In terms of performance, i.e., delay (Fig. 5) and number of cores (Fig. 6), EvoVNFP is roughly the same as RandImm and both are lower than conventional EA. This means that EvoVNFP does not sacrifice the performance in order to adapt to the dynamics.

2) Evaluation of different period lengths: Next we compare EvoVNFP at different congestion levels of the system represented by different period lengths T_p in Fig. 7 and Fig. 8. The period is the interval of smallest fluctuation of the objective. The x-axes represent the T_p values in generations ($T_p = 1, 3, 5, 7, 10, 20, 50, 100, 200$) and the y-axes illustrate failure probability and the number of generations until the first feasible solution, respectively. In these simulations, we set the average inter-arrival time to $E[T_A] = 1000$ and sojourn time to $E[T_B] = 5000$ generations. In the legend of Fig. 7 and Fig. 8, we denote by R the number of the chains in the system.

In Fig. 7, the failure probability at $T_p = 5$ in $R = 3$ and $R = 5$ becomes smallest among all values of T_p. However, the result at $T_p = 5$ in $R = 1$ is not minimal in Fig. 7, but it is smallest in Fig. 8. This is because it is easy to generate solutions when R is small. Then EvoVNFP can quickly obtain solutions at all T_p and therefore Fig. 7 does not show this difference. However, the speed to evolve the individual is faster at $T_p = 5$ than for the other T_p, which is illustrated in Fig. 8. In conclusion from these two observations, $T_p = 5$ is the best setting for the considered traffic conditions.

3) Evaluation for different degrees of mutation: Finally, we compare EvoVNFP at different system loads using different degrees of mutation in Fig. 9 and Fig. 10. In normal EA, solutions are obtained faster when the number of mutations becomes large. However, when there are too many mutations, an EA step approaches a random search in the solution space and the performance becomes lower. We define the magnitude of a mutation as the number of elements that mutation can change in one individual. In this study, we realized the difference of the magnitude of a mutation by repeating the mutation operation explained in Section III-B3. The x-axes in Fig. 9 and Fig. 10 represent the number of repeated mutations in one iteration step and the y-axes represent the metrics of failure probability and number of generations until first feasible solution, respectively. We consider that in a single generation there may be 1, 3, 5, 7, or 9 mutations and we again define R as the number of chains in the system. In these simulations, we set $E[T_A] = 1000$, $E[T_B] = 5000$, and $T_p = 20$ generations.

Figure 9 shows that the smaller the number of mutations per step are, the lower the failure probability is. This is because these small fluctuations of objectives can be regarded as another mechanism of searching. On the other hand, Fig. 10 shows that the best results are achieved for 3 mutations. However, we consider that a smaller number of repetitions is preferential because Fig. 10 is the conditional result depending on the condition in Fig. 9. These results indicate that EvoVNFP does not need a large number of mutations to achieve a good performance.

V. Conclusion and Future Work

In this paper, we evaluated EvoVNFP under different loads and parameter settings to discuss its adaptability under various conditions. The results of the evaluations showed that EvoVNFP is better than the two reference methods in terms of the adaptability without sacrificing the quality of the generated placements. Moreover, the results show that there are specific parameter settings of EvoVNFP which make its adaptability even better; the lengths of the periods for goal fluctuation should be about 5 generations and the magnitude of mutations should be kept small. Our future work includes

determining further metrics for comparing EvoVNFP with other methods to demonstrate its good performance.

ACKNOWLEDGMENT

This research was supported in part by the "Program for Leading Graduate School" of the Ministry of Education, Culture, Sports, Science and Technology in 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network Function Placement for NFV Chaining in Packet/Optical Datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, Apr. 2015.

[8] 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.

[9] 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.

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

[11] 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.

[12] M. Otokura, K. Leibnitz, Y. Koizumi, D. Kominami, T. Shimokawa, and M. Murata, "Application of Evolutionary Mechanism to Dynamic Virtual Network Function Placement," in *Proceedings of ICNP Workshop on Control Operation and Application in SDN protocols (CoolSDN)*, Nov. 2016.

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

[14] 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.

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

[16] 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.