Applied Soft Computing 112 (2021) 107798

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A two-stage approach for multicast-oriented virtual network function placement

Xinhan Wang, Huanlai Xing *, Dawei Zhan, Shouxi Luo, Penglin Dai, Muhammad Azhar Iqbal

School of Computing and Artificial Intelligence, Southwest Jiaotong University, China

ARTICLE INFO

Article history:
Received 6 August 2020
Received in revised form 30 May 2021
Accepted 9 August 2021
Available online 12 August 2021

Keywords:
Estimation of distribution algorithm
Multicast
Network function virtualization
Virtual network function placement

ABSTRACT

Network function virtualization (NFV) is an emerging network paradigm that decouples softwarized network functions from proprietary hardware. Nowadays, resource allocation has become one of the hot topics in the NFV domain. In this paper, we formulate a service function chain (SFC) mapping problem in the context of multicast, which is also referred to as the multicast-oriented virtual network function placement (MVNFP) problem. The objective function considers end-to-end delay as well as compute resource consumption, with bandwidth requirements met. A two-stage approach is proposed to address this problem. In the first stage, Dijkstra's algorithm is adopted to construct a multicast tree. In the second stage, a novel estimation of distribution algorithm (nEDA) is developed to map a given SFC over the multicast tree. Simulation results show that the proposed two-stage approach outperforms a number of state-of-the-art evolutionary, approximation, and heuristic algorithms, in terms of the solution quality.

1. Introduction

Multicast is an efficient point-to-multipoint data delivery mode in computer networks, which duplicates packets at intermediate nodes if necessary and sends the copies to different destinations simultaneously. Compared with multiple unicast sessions, a multicast session can significantly reduce bandwidth consumption [1]. Hence, this mode has great potential to support ever-increasing multimedia applications [2], such as video conferencing, distance education, IPTV, interactive games, and so on. Current multicast services always require data flows to traverse specific network facilities (also called middle-boxes) before arriving at destinations. These middle-boxes are proprietary hardware fixed somewhere in the network, which leads to slow service deployment and network rigidity.

Network function virtualization (NFV) has been envisioned as one of the promising paradigms for future network service provision [3]. NFV decouples network functions, e.g., Firewall, network address transfer (NAT), and intrusion detection system (IDS), from proprietary hardware and implements them as virtual network functions (VNFs). Placing VNFs on compute nodes not only significantly improves network flexibility but also greatly reduces both capital expenditure (CAPEX) and operational expense (OPEX) from the point of view of network service providers (NSPs) [4].

In NFV, a service function chain (SFC) is a set of ordered VNFs. If a data flow requests a specific network service, it must be processed by all VNFs in a given SFC in the correct order before it reaches a destination. VNF placement (VNFP) is to map an SFC over a substrate network to deploy an explicit network service, which is also known as SFC mapping. The VNFP problem is one of the most challenging issues in NFV resource allocation [5]. An appropriate VNFP solution is key to high-quality network services. An example of VNFP for a unicast session is shown in Fig. 1, where an SFC is to be deployed on a number of compute nodes in a physical network. This SFC contains three VNFs, namely *VNF*1, *VNF*2, and *VNF*3. The data-flow from *Source* to *Destination* must be processed by *VNF*1, *VNF*2, and *VNF*3 one by one. In the VNFP process, compute nodes, *A*, *B*, and *D*, are selected to host the three VNFs above. Therefore, the data-flow generated from *Source* needs to visit nodes $A \rightarrow B \rightarrow D$ before it arrives at *Destination*.

In the area of VNFP, mainstream research focuses on unicast communications [6,7]. VNFP in multicast, is not trivial at all since multicast has been identified as an important supporting technique for next-generation computer networks. Increasingly more applications supported by multicast are to be witnessed in the future. Compared with unicast, VNFP in multicast is much more complicated in terms of problem modeling and solving [8]. First of all, instead of considering a single path between a source

* Correspondence to: Room 9439, Teaching building 9, School of Computing and Artificial Intelligence, Southwest Jiaotong University (Xipu Campus), Chengdu 611756, China.
E-mail address: hxx@home.swjtu.edu.cn (H. Xing).

https://doi.org/10.1016/j.asoc.2021.107798
1568-4946/

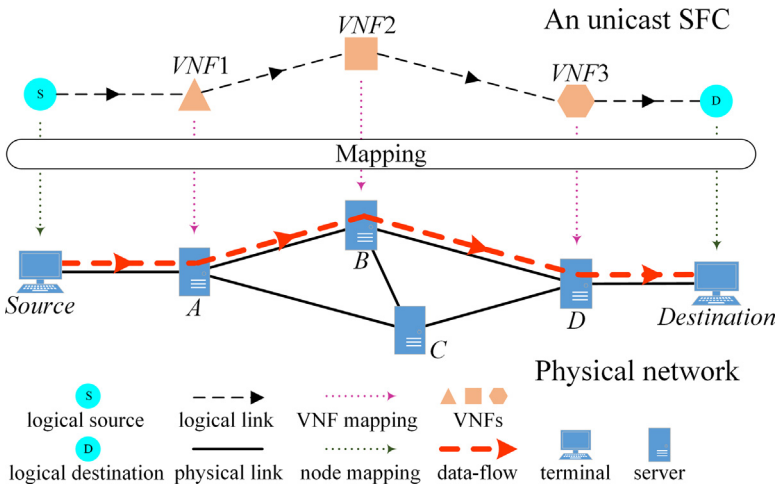

Fig. 1. An example of VNFP in unicast.

List of the main abbreviations

2DPSSE	2-dimensional problem-specific solution encoding
ABF	Average best fitness
ACT	Average computational time
CAPEX	Capital expenditure
EA	Evolutionary algorithm
EDA	Estimation of distribution algorithm
FPVU	Flexible probability vector update
GA	Genetic algorithm
ILP	Integer linear programming
INPLD-PV	Inner-path location dependency PV
IPL-PV	Initial placement location PV
ITPLD-PV	Inter-path location dependency PV
MILP	Mixed integer linear programming
MVNFP	Multicast-oriented VNF placement
nEDA	Novel estimation of distribution algorithm
NFV	Network function virtualization
NSP	Network service provider
OPEX	Operational expense
PSO	Particle swarm optimization
PV	Probability vector
QoS	Quality of service
SDN	Software defined networking
SFC	Service function chain
TS-M	Two-stage approach for the MVNFP problem
VNF	Virtual network function
VNFP	VNF placement

and destination pair, VNFP in multicast takes a multicast tree into account, which consists of multiple paths that originate from an identical source and terminate at different destinations. Secondly, an identical SFC is to be mapped along each path in the multicast tree, so that data-flow along any path is processed by all VNFs in the SFC in the right order before the flow arrives at a destination. Last but not least, as some paths of a multicast tree may overlap partially, the overlapped parts can actually share VNFs with the same functionality. Fig. 2 shows an example of VNFP in multicast.

Hosting a VNF incurs physical resource consumption on a compute node, for example, computing, storage, and buffering. Hosting more VNFs leads to heavier physical resource consumption. Therefore, when mapping SFC for multicast, it is of vital importance to minimize the physical resource consumption while meeting various constraints, such as bandwidth and computing capacities. This problem is referred to as the multicast-oriented VNF placement (MVNFP) problem, which is NP-hard [9].

In order to save CAPEX and OPEX costs, NSPs are interested in maximizing the resource utilization of infrastructure while minimizing the investment in new network elements. Reducing the physical resource consumption in a network is an essential demand from NSPs [10]. Meanwhile, quality of service (QoS) is a key concern from the user perspective [11]. NSPs need to consider QoS requirements to guarantee user experiences, such as delay and bandwidth. Both physical resource consumption and QoS are equally important. However, only a few works consider them simultaneously when studying the MVNFP problem. For example, a heuristic algorithm was developed to minimize the link cost and the compute resource consumption [12]. However, the bandwidth consumption was ignored. In [13], an approximation algorithm was proposed to solve the resource allocation problem in multicast. The authors tried to minimize the total setup and connection cost, representing the total cost of VNFs and links. Nevertheless, the end-to-end delay was not considered. Besides, in [14], the physical resource consumption and QoS were taken into account in the multicast routing problem in the context of NFV. The compute resource consumption, the end-to-end delay, and the bandwidth consumption were all considered. However, they were treated differently, i.e., the compute resource consumption was prioritized while the other two were secondary concerns. On the one hand, the compute resource consumption is, to a certain extent, able to reflect the NSPs' demand [15]. On the other hand, the end-to-end delay and bandwidth are two commonly used QoS parameters reflecting user experience [14]. The three issues should be considered at the same time as they stand for the benefits that NSPs and end-users are interested in. Therefore, it is natural to study the MVNFP problem with the compute resource consumption, the end-to-end delay as well as bandwidth consumption considered.

Unlike conventional mathematical deduction approaches, e.g., linear and dynamic programming, evolutionary algorithms (EAs) are a family of nature-inspired global search and optimization methods with strong robustness, fast convergence, and wide applications. EAs have been considered one of the ideal candidate techniques for handling NP-hard problems since they

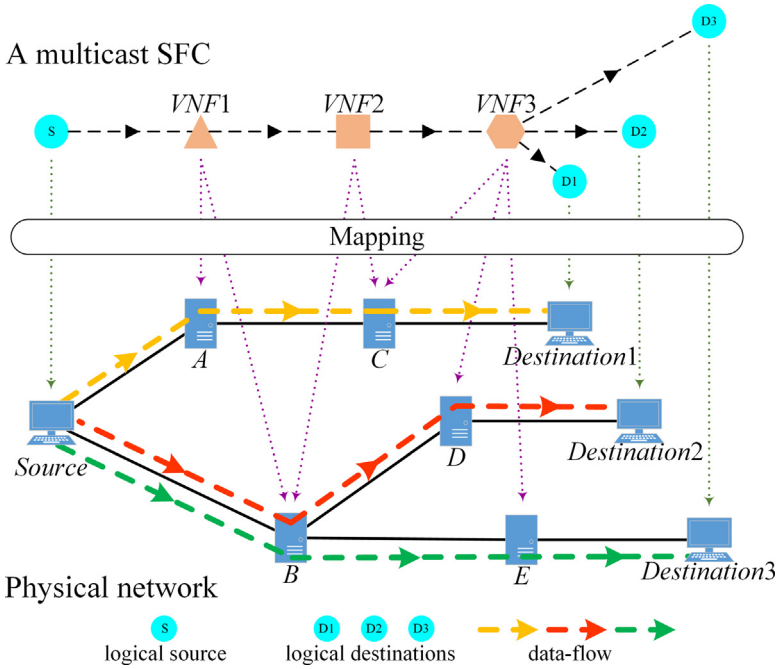

Fig. 2. An example of VNFP in multicast.

can output an acceptable solution to a given problem within a limited computational time. Estimation of distribution algorithm (EDA) is a branch of EAs combining evolutionary computation and machine learning [16]. As a stochastic optimization algorithm, EDA explores search space by sampling one or more explicit probabilistic models built based on promising solutions found during evolution [17]. EDA has been widely applied to a variety of areas, including scheduling problem [18,19], vehicle routing problem [20], engineering optimization [21], multiobjective optimization [22], military development [23], environmental science [24], machine learning [25], network systems [26], and so on.

This paper formulates a variant of the MVNFP problem consisting of two closely correlated sub-problems. It presents a two-stage approach to address the problem. Our main contributions are summarized below.

(1) The MVNFP problem is composed of a multicast tree construction sub-problem and a VNFP sub-problem. The end-to-end delay and compute resource consumption are both considered in the objective function, and the bandwidth consumption is constrained. A two-stage approach for the MVNFP problem (TS-M) is designed to address the two sub-problems above. In the first stage, we adopt Dijkstra's algorithm to construct a multicast tree for a given multicast service request. In the second stage, we propose a novel EDA (nEDA) to place VNFs over the multicast tree constructed.

(2) The proposed nEDA is featured with three performance-enhancing schemes, including a 2-dimensional problem-specific solution encoding (2DPSSE) scheme, a three-probability-vector-based solution generation scheme, and a flexible probability vector update (FPVU) scheme. In the first scheme, a solution to VNFP is represented by a 2D vector, where each row in the vector denotes a path from the source to one destination. In the second scheme, nEDA maintains three probability vectors for generating all locations for hosting VNFs based on the multicast tree. In the third one, an elitist solution set is adopted to update the three probability vectors above.

(3) 18 test instances are used for algorithm evaluation. The simulation results demonstrate that nEDA outperforms five widely used evolutionary algorithms in terms of the VNFP solution quality. In addition, the proposed TS-M obtains the lowest cost compared with five state-of-the-art approximation and heuristic algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related work. Problem formulation is described in Section 3. Section 4 presents the two-stage approach to the newly modeled MVNFP problem, including the proposed nEDA. Section 5 discusses the experimental results. Finally, Section 6 concludes the paper and presents future work.

2. Related work

The MVNFP problem has received more and more research attention from both academia and industry. The mainstream research defines MVNFP problems as mathematical models based on integer linear programming (ILP) and mixed integer linear programming (MILP) and develops heuristics to tackle them. For example, Ren et al. [12] formulated the optimal service function tree embedding problem based on ILP, where a two-stage algorithm with an approximation ratio of $1+\rho$ was developed. Kiji et al. [27] studied the VNFP and routing problem and designed a multicast service chaining model based on ILP that merges multiple service paths, where the VNFP cost and link usage were taken into account. Qin et al. [28] were concerned with enabling multicast slices in edge networks. They formulated the delay-aware network slicing problem into the ILP model and proposed an approximation algorithm to minimize the implementation cost. Muhammad et al. [2] considered multisource multicast resource optimization in NFV. They proposed two heuristic algorithms based on the MILP model, aiming to minimize bandwidth and CPU resource consumption. Zhang et al. [29] studied the multicast routing problem in the NFV and software defined networking (SDN) environment, focusing on static and dynamic multicast tree construction. They formulated the problem based on MILP and adopted a branch and bound method to address it. Later on, the

authors developed similar algorithms to reduce the computational cost [30,31]. In [32], Gao et al. studied how to map virtual networks to physical networks for supporting reliable multicast services. They proposed a MILP model with max–min fairness reliability and tackled it by a genetic algorithm (GA) with uniform reliability mutation. Zeng et al. [33] considered how to orchestrate multicast-oriented NFV trees in an inter-datacenter optical network and presented a path-intersection heuristic based on MILP. Alhussein et al. [34] took multipath traffic routing between embedded VNFs into account. They devised a heuristic algorithm based on the MILP model, aiming at minimizing the provisioning cost of both VNFs and links. Qu et al. [35] studied the problem of provisioning multi-source multicast services in softwarized networks. A K-shortest path-based greedy algorithm based on the MILP model was proposed to determine multi-source multicast hybrid routing.

Other models and methods have received increasingly more research efforts. For instance, Kuo et al. [13] studied the allocation of VNFs in SDN multicast routing, where an approximation algorithm, named service overlay forest, was proposed to minimize the total cost of all allocated VNFs and multicast trees. Xie et al. [14] considered the multi-source multicast routing problem with QoS parameters constrained, where a heuristic algorithm was developed to improve resource utilization. Xu et al. [36] proposed the concept of the pseudo-multicast tree, where the same network functionality could be deployed only once. Some data-flows had to travel longer distances before arriving at their destinations than others, which could cause larger end-to-end delay and thus deteriorate user experience. Then, the authors addressed the resource sharing problem of NFV-enabled multicasting in mobile edge cloud networks by an approximation algorithm [37]. In [38] and [39], Yi et al. proposed a multi-stage solution in hybrid infrastructure with VNFs and network functions. They adopted a minimum spanning tree method to construct the traffic forwarding topology and used a backtracking strategy for network function delivery. Ma et al. [8] studied the NFV-enabled multicast problem in mobile edge computing networks with static and dynamic multicast requests and proposed several approximation algorithms to maximize network throughput. Soni et al. [40] were concerned with the NFV-based multicast service problem in software defined ISP networks, where a lazy load balancing multicast routing algorithm was devised to strike a balance between the guaranteed-bandwidth multicast traffic and the best-effort traffic. Cheng et al. [41] proposed a heuristic approach based on the Steiner tree with all destination nodes as terminals. This approach is to reduce the cost of VNFP and routing. Zahoor et al. [42] proposed an overlay multicast network architecture for an ad-hoc multicast service. They developed several algorithms to minimize the bandwidth utilization, radio access network resources, and cost.

As aforementioned, NFV is an emerging yet prospective research area. However, very limited research effort has been dedicated to the study on the MVNFP problem. From the point of view of problem formulation, end-to-end delay and compute resource consumption are both important and usually regarded as separate objectives for minimization. But, little attention has been paid to considering them simultaneously. This motivates us to model the MVNFP problem with the two aspects taken into account. From the point of view of problem-solving, approximation and heuristic algorithms are the mainstream focus, which usually leads to unstable performance with respect to the solution quality. On the other hand, EAs aim at searching for global optima in the search space and usually obtain better solutions than approximation and heuristic algorithms. In particular, EDA has shown its great potential in addressing various engineering problems. Yet, to the best of our knowledge, it has not been applied to multicast-oriented VNFP. This inspires us to develop a novel EDA to tackle the newly modeled MVNFP problem.

3. Problem formulation

This section introduces the system model and notations first. Then, we define the problems precisely, including the multicast tree construction and the VNF placement sub-problems, respectively. The main notations used in problem formulation are summarized in Table 1.

3.1. System model

Let graph $G = (V, E)$ represents a communications network, where $V = v_1, v_2, \ldots, v_{|V|}$ and $E = e_1, e_2, \ldots, e_{|E|}$ are the node and link sets, respectively. $|V|$ and $|E|$ are the cardinalities of V and E, respectively. Each node $v \in V$ is able to host VNF(s) and forward data-flows. A node with no VNF hosted simply forwards its incoming data-flow to an outgoing link. The available compute resource of node v is denoted by R_v. Transmitting data over a physical link incurs propagation delay. For an arbitrary link $e \in E$, we denote its associated propagation delay by $D_{prpg}(e)$. The available bandwidth of link e is represented by $B(e)$.

In order to maximize the utilization of each VNF, this paper does not allow placing two or more VNFs with the same functionality on a single node, which indicates any two VNFs on the same node offer different functions. In a multicast tree, different paths may have some common nodes. VNFs placed on these nodes can be shared by the corresponding paths [37,43]. Take Fig. 2 as an example, *VNF*1 and *VNF*2 hosted by node B are shared by two paths, namely *Source* $\rightarrow B \rightarrow D \rightarrow$ *Destination*2 and *Source* $\rightarrow B \rightarrow E \rightarrow$ *Destination*3.

3.2. Multicast service request

A multicast service request is expressed as a 4-tuple set $MSR = \{s, T, F, BW\}$. s is the source node. $T = \{t_1, t_2, \ldots, t_{|T|}\}$ is the set of destination nodes, where $|T|$ is the cardinality of T. $F = f_1, f_2, \ldots, f_{|F|}$ is the set of ordered VNFs, namely the requested SFC, where f_j is the jth VNF in F, $j = 1, 2, \ldots, |F|$, and $|F|$ is the cardinality of F. The compute resource consumed for hosting $f \in F$ is denoted by R_f. $BW = \{bw_{1,2}, bw_{2,3}, \ldots, bw_{|F|-1,|F|}\}$ is the set of bandwidth requirements [44], where $bw_{j,j+1}$ is the bandwidth requirement between f_j and $f_{j+1}, j = 1, 2, \ldots, |F| - 1$.

3.3. Problem definitions

Given a multicast service request MSR, we divide the MVNFP problem into two sub-problems: multicast tree construction and VNFP. The first sub-problem aims at constructing a multicast tree connecting s and all destinations in T. The second sub-problem aims at placing all VNFs in F on the constructed multicast tree so that data-flow along any path is processed by $f_1, f_2, \ldots, f_{|F|}$ one by one before it reaches a destination in T.

3.3.1. Multicast tree construction sub-problem

As aforementioned, a multicast tree consists of multiple paths, each originating from s and terminating at a destination. Let $G_M = (V_M, E_M)$ represent a multicast tree constructed for hosting a given SFC, where $G_M \subseteq G$, $V_M \subseteq V$, and $E_M \subseteq E$. Let $Path(i)$ be the path from source s to destination $t_i \in T$, $i = 1, 2, \ldots, |T|$, in G_M. Let $V^i = v^i_1, v^i_2, \ldots, v^i_{|V^i|}$ and $E^i = e^i_1, e^i_2, \ldots, e^i_{|E^i|}$ be the node and link sets of $Path(i)$, respectively, where $|V^i|$ and $|E^i|$ are the cardinalities of V^i and E^i, respectively.

We denote the total propagation delay along $Path(i)$ before mapping SFC F by $D^{bfr}_{prpg}(Path(i))$, $i = 1, 2, \ldots, |T|$, as defined in Eq. (1).

$$D^{bfr}_{prpg}(Path(i)) = \sum_{e \in E^i} D_{prpg}(e) \tag{1}$$

Table 1
Summary of the main notations in problem formulation.

Notation	Definition	Notation	Definition
$B(e)$	Available bandwidth of link e	$E^i_{\|F\|\to t}$	Link set of the sub-path from the node hosting $f_{\|F\|}$ to destination t_i along $Path(i)$
BW	Set of bandwidth requirements	F	Set of ordered VNFs
$bw_{j,j+1}$	Bandwidth requirement between VNFs f_j and f_{j+1}	G	Communications network
C_D	Cost of the average end-to-end delay along all paths in G_M	G_M	Multicast tree constructed for hosting a given SFC
C_R	Cost of the compute resource consumed for hosting F over G_M	MSR	Multicast service request
$D_{prcs}(v^i_k, f_j)$	Processing delay incurred if a data-flow is processed by VNF f_j, where f_j is hosted by node v^i_k	$Path(i)$	Path from source s to destination t_i
$D_{prpg}(e)$	Propagation delay of link e	R_f	Compute resource consumed for hosting VNF f
$D^{aft}_{prpg}(Path(i))$	The total propagation delay along $Path(i)$ after all VNFs in SFC F are placed	R_v	Available compute resource of node v
$D^{bfr}_{prpg}(Path(i))$	The total propagation delay along $Path(i)$ before mapping SFC F	s	Source node
E	Link set of G	T	Set of destination nodes
E_M	Link set of G_M	$U(v)$	Set of VNFs placed on node v
E^i	Link set of $Path(i)$	V	Node set of G
$E^i_{s\to 1}$	Link set of the sub-path from s to the node hosting f_1 along $Path(i)$	V_M	Node set of G_M
$E^i_{j\to j+1}$	Link set of the sub-path between the nodes hosting f_j and f_{j+1} along $Path(i)$	V^i	Node set of $Path(i)$

The multicast tree construction sub-problem is to construct a multicast tree $G_M = (V_M, E_M)$ from G, with the bandwidth constraint satisfied [43]. The objective is to minimize the average total propagation delay along all paths, as expressed in Eq. (2). For any $e \in E_M$, $B(e)$ must be at least not less than the maximum value among $bw_{1,2}, bw_{2,3}, \ldots, bw_{|F|-1,|F|}$, and a basic bandwidth requirement is defined in Inequality (3).

$$Minimize: \quad \frac{1}{|T|}\sum_{i=1}^{|T|} D^{bfr}_{prpg}(Path(i)) \tag{2}$$

$$Subject\ to: B(e) \geq \max\{bw_{1,2}, bw_{2,3}, \ldots, bw_{|F|-1,|F|}\}, \forall e \in E_M \tag{3}$$

3.3.2. VNF placement sub-problem

The task of VNFP is to map a given SFC F over the constructed multicast tree G_M, where G_M is composed of $Path(i)$, $i = 1, 2, \ldots, |T|$. To be specific, we need to place $\{f_1, f_2, \ldots, f_{|F|}\}$ along each path in G_M so that for an arbitrary path, its associated data-flow is processed by $f_1 \to f_2 \to \ldots \to f_{|F|}$ before it arrives at its destination [43]. We denote the set of VNFs placed on node $v \in V_M$ by $U(v)$, where $U(v) \subseteq F$. The relationship between $U(v)$ and F is defined in Eq. (4).

$$\bigcup_{v\in V^i} U(v) = \{f_1, f_2, \ldots, f_{|F|}\} = F \tag{4}$$

For an arbitrary node $v \in V_M$, its available compute must sufficient for hosting all the VNFs placed on v. Thus, the relationship between R_v and R_f is constrained by Inequality (5).

$$R_v \geq \sum_{f\in U(v)} R_f, v \in V_M \tag{5}$$

Note that as inappropriate VNFP may happen, locations for hosting VNFs might be in the wrong order. This could cause a data-flow to be transmitted forward and backward over the same link more than once. Hence, $B(e)$ and BW are constrained by a tighter bandwidth requirement, written in Inequality (6).

$$B(e) \geq \sum_{j=1}^{|F|-1} \left(\mu^e_{j,j+1} \cdot bw_{j,j+1}\right), \forall e \in E_M \tag{6}$$

where $\mu^e_{j,j+1} \in \{0, 1\}$ is a binary variable indicating if e is used to transmit a data-flow from f_j to f_{j+1}. We have $\mu^e_{j,j+1} = 1$ if a data-flow from f_j to f_{j+1} is transmitted over e and $\mu^e_{j,j+1} = 0$, otherwise. Fig. 3 shows an example of bandwidth consumption in an inappropriate VNFP scenario. Note that, each number in data-flow stands for the number of VNFs that the data-flow has already gone through. For example, "2" indicates the data-flow has been processed by two VNFs, namely $VNF1$ and $VNF2$. Due to the inappropriate locations chosen for hosting $VNF2$ and $VNF3$, data-flow is transmitted over link e_3 forward and backward multiple times. Hence, $B(e_3)$ should be at least not less than the summation of $bw_{1,2}$, $bw_{2,3}$ and $bw_{3,4}$.

Let $D^{aft}_{prpg}(Path(i))$ be the total propagation delay along $Path(i)$ after all VNFs in SFC F are placed, $i = 1, 2, \ldots, |T|$, as defined in

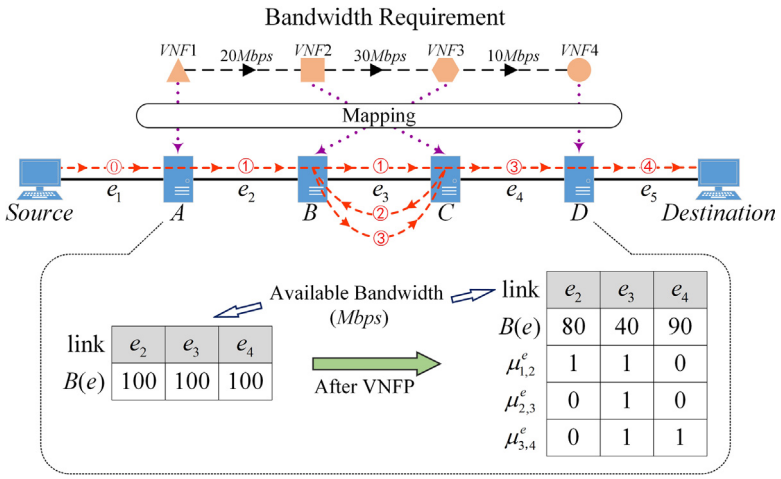

Fig. 3. An example of bandwidth consumption.

Eq. (7).

$$D^{aft}_{prpg}(Path(i)) = \sum_{e\in E^i_{s\to 1}} D_{prpg}(e) + \sum_{j=1}^{|F|-1} \sum_{e\in E^i_{j\to j+1}} D_{prpg}(e) + \sum_{e\in E^i_{|F|\to t}} D_{prpg}(e) \tag{7}$$

where, $E^i_{s\to 1} \subset E^i$ is the link set of the sub-path from s to the node hosting f_1 along $Path(i)$, $E^i_{j\to j+1} \subset E^i$ is the link set of the sub-path between the nodes hosting f_j and f_{j+1} along $Path(i)$, and $E^i_{|F|\to t} \subset E^i$ is the link set of the sub-path from the node hosting $f_{|F|}$ to destination t_i along $Path(i)$, respectively. Note that $D^{aft}_{prpg}(Path(i))$ may not be the same as $D^{bfr}_{prpg}(Path(i))$ because mapping F over $Path(i)$ might change the order of nodes to be visited by the corresponding data-flow. Take Fig. 3 as an example, before the VNFP, the order of nodes the data-flow visits is $A\to B\to C\to D$. In contrast, after the VNFP, the order is changed to $A\to B\to C\to B\to C\to D$, which obviously leads to additional delay.

A node hosting VNF(s) incurs a processing delay since an incoming data-flow is processed by one or more VNFs before it is forwarded. Let $D_{prcs}(v^i_k, f_j)$, $i = 1, 2, \ldots, |T|$, $j = 1, 2, \ldots, |F|$, $k = 1, 2, \ldots, |V^i|$, denote the processing delay incurred if a data-flow is processed by VNF f_j, where f_j is hosted by node $v^i_k \in V^i$. Meanwhile, a node also incurs forwarding delay if it needs to forward some incoming data-flows. However, compared with processing delay, forwarding delay is trivial. So, this paper ignores the forwarding delay incurred on each node. For an arbitrary node $v^i_k \in V^i$ with VNF(s) placed, its associated processing delay is the summation of $D_{prcs}(v^i_k, f_j)$, as long as f_j is hosted by v^i_k.

The end-to-end delay along $Path(i)$, $i = 1, 2, \ldots, |T|$, is composed of the total propagation delay, $D^{aft}_{prpg}(Path(i))$, and the summation of $D_{prcs}(v^i_k, f_j)$, $v^i_k \in V^i$, $j = 1, 2, \ldots, |F|$. Let C_D be the cost of the average end-to-end delay along all paths in G_M, as defined Sin Eq. (8).

$$C_D = \frac{1}{|T|}\sum_{i=1}^{|T|}\left(D^{aft}_{prpg}(Path(i)) + \sum_{k=1}^{|V^i|}\sum_{f_j\in U(v^i_k)} \mu^i_{k,j}\cdot D_{prcs}(v^i_k, f_j)\right) \tag{8}$$

where $\mu^i_{k,j} \in \{0, 1\}$ is a binary variable indicating if f_j hosted by v^i_k is used to process the data-flow along $Path(i)$. We have $\mu^i_{k,j} = 1$ if f_j is hosted by v^i_k and responsible for processing the data-flow along $Path(i)$, and $\mu^i_{k,j} = 0$ otherwise.

Let C_R be the cost of the compute resource consumed for hosting F over G_M, as defined in Eq. (9).

$$C_R = \sum_{v\in V_M}\sum_{f\in U(v)} R_f \tag{9}$$

In the VNFP sub-problem, our task is to map a given SFC over a multicast tree constructed in Section 3.2, with the total cost regarding the end-to-end delay and the compute resource consumption minimized and all constraints met. The total cost is defined in Eq. (10).

$$Minimize: \quad C_D + \alpha\cdot C_R \tag{10}$$

$$Subject\ to: \quad \text{Eq. (4), Inequalities (5) and (6)} \tag{11}$$

where $\alpha \in (0, 1)$ is a weight reflecting the importance of C_R against C_D. Eq. (4) ensures that for any $Path(i)$, the corresponding data-flow can be processed by all VNFs in F, $i = 1, 2, \ldots, |T|$. Inequality (5) means that when hosting VNFs, any node in the multicast tree cannot exceed its available compute resource. Inequality (6) guarantees that for any link $e \in E_M$, the total bandwidth consumption caused by a data-flow cannot exceed the available bandwidth $B(e)$, if this data-flow is transmitted over e multiple times due to the inappropriate locations selected for hosting VNFs.

4. A two-stage approach to the MVNFP problem

As aforementioned, the MVNFP problem is composed of the multicast tree construction sub-problem and the VNFP sub-problem. So, we propose a two-stage approach to tackle the problem. In the first stage, we use Dijkstra's algorithm [45], a well-known shortest path algorithm, to find a multicast tree with the average total propagation delay minimized and the basic bandwidth constraint met. In the second stage, we present a novel EDA, namely nEDA, to place the required VNFs over the obtained multicast tree, minimizing the compute resource consumption and the average end-to-end delay and satisfying the tighter bandwidth constraint. The main notations used in the proposed two-stage approach are summarized in Table 2.

4.1. Dijkstra's algorithm for multicast tree construction

When a multicast service request $MSR = \{s, T, F, BW\}$ arrives, Dijkstra's algorithm is applied to construct a multicast tree $G_M = (V_M, E_M)$ [46]. The procedure for constructing a multicast tree is shown in Algorithm 1, where for each link, its associated propagation delay is used as its weight. Fig. 4 illustrates an example

Table 2
Summary of the main notations in the proposed two-stage approach.

Notation	Definition	Notation	Definition
ES	Set of elitist solutions	$\boldsymbol{p}_i^{IT}$	Probability matrix containing all probabilistic information about selecting location $loc_{i,1}$ to host f_1
$loc_{i,j}^k$	ID of the node where $f_j \in F$ is to be placed in $Path(i)$	$\rho_{m,n}^{IT(i)}$	Probability of selecting the nth node in $Path(i)$ to host f_1, given that f_1 is placed on the mth node in $Path(i-1)$
N_{pop}	Population size	$Path(max)$	The longest path among all paths
N_{ES}	Number of solutions in ES	$\lvert V^{max}\rvert$	Number of nodes in $Path(max)$
N_{iter}	Number of iterations	$\boldsymbol{W}^{IN}$	Weight set for updating INPLD-PV
$\boldsymbol{P}^{IN}$	INPLD-PV	$\omega_n^{IN(i,j,m)}$	Number of the nth node in $Path(i)$ selected to host f_j, given that f_{j-1} is placed on the mth node in $Path(i)$
$\boldsymbol{p}_{i,j}^{IN}$	Probability matrix containing all probabilistic information about selecting all locations along $Path(i)$ for hosting $f_2, f_3, \ldots, f_{\lvert F\rvert}$	$\boldsymbol{W}^{IP}$	Weight set for updating IPL-PV
$\rho_{m,n}^{IN(i,j)}$	Probability of selecting the nth node in $Path(i)$ to host f_j, given that f_{j-1} is placed on the mth node in $Path(i)$	w_k^{IP}	Number of the kth node in $Path(1)$ selected to host f_1
$\boldsymbol{P}^{IP}$	IPL-PV	$\boldsymbol{W}^{IT}$	Weight set for updating ITPLD-PV
p_k^{IP}	Probability of selecting the kth node of V^1 to host f_1	$\omega_n^{IT(i,m)}$	Number of the nth node in $Path(i)$ selected to host f_1, given that f_1 is placed on the mth node in $Path(i-1)$
$\boldsymbol{P}^{IT}$	ITPLD-PV	Y_k	A solution to VNFP in G_M

network and its resultant multicast tree, where all selected links are in bold.

Algorithm 1 Dijkstra's algorithm

Input: $G = (V, E)$ and $MSR = \{s, T, F, BW\}$.
Output: $G_M = (V_M, E_M)$.

```
Set V_M = ∅ and E_M = ∅;
for e ∈ E do
    if B(e) is no less than the maximum value of BW do  // see Inequality (3)
        Remove e from E;
for each destination node t_i in T do
    Use Dijkstra's algorithm to obtain the shortest path Path(i);
    Set V_M = V_M ∪ V^i;      // add all nodes in Path(i) to V_M
    Set E_M = E_M ∪ E^i;      // add all links in Path(i) to E_M
Obtain G_M = (V_M, E_M) as the multicast tree;
```

4.2. nEDA for VNF placement

Instead of evolving an explicit population, an EDA maintains one or more probability vectors (PVs) that estimate the distribution of promising solutions [17]. With the evolution continuing, these PVs generate fitter solutions with higher probabilities [47]. Statistical information is extracted from promising samples and used to update PVs. EDA usually achieves better optimization performance than GA, particle swarm optimization (PSO), and greedy approaches when tackling large and complex optimization problems such as network coding [47], feature selection [48], mission planning [49], path planning [50], and so on. In addition, EDA has been successfully applied to the VNFP problem in unicast, indicating its potential in addressing the MVNFP problem [51].

This paper proposes a three-feature nEDA to handle the problem above. The first feature is a 2-dimensional problem-specific solution encoding (2DPSSE) scheme. The second one is to jointly use three probability vectors for placing VNFs along all paths within the constructed multicast tree. The last one is a flexible probability vector update (FPVU) scheme to avoid prematurity.

This section introduces the solution representation based on 2DPSSE first and the three probability vectors later. After that, the FPVU scheme is described. In the end, the pseudo-code of nEDA is given.

4.2.1. Solution representation and fitness evaluation

As VNFP in multicast is an emerging topic in NFV, addressing it by EAs has not attracted enough research attention. To our best, GA was the only one adapted for a MILP-based MVNFP problem with max–min fairness reliability, where a 1-dimensional encoding scheme was used to represent solutions [32]. However, this encoding maps source and destinations only, with detailed SFC mapping ignored. There are two drawbacks. Firstly, it cannot clearly reflect the locations where all VNFs are placed. This usually leads to inefficient evolution due to the indirect representation. Secondly, infeasible solutions are generated at high probability, even for small network topology. Hence, the 1-dimensional encoding is not suitable for large-scale networks. This motivates us to develop a more effective solution representation, namely the 2DPSSE scheme.

Assume there is a multicast service request $MSR = \{s, T, F, BW\}$, where $T = \{t_1, t_2, \ldots, t_{|T|}\}$ and $F = \{f_1, f_2, \ldots, f_{|F|}\}$. Let N_{pop} be the population size. In the 2DPSSE scheme, an arbitrary solution to VNFP in G_M, Y_k can be represented by a 2D vector, as shown in Eq. (12), $k = 1, 2, \ldots, N_{pop}$.

$$Y_k = \begin{bmatrix} loc_{1,1}^k & \cdots & loc_{1,|F|}^k \\ \vdots & \ddots & \vdots \\ loc_{|T|,1}^k & \cdots & loc_{|T|,|F|}^k \end{bmatrix} \tag{12}$$

Recall that $Path(i)$ is the path from source s to destination $t_i \in T$, $i = 1, 2, \ldots, |T|$. The ith row in Y_k stands for all locations along $Path(i)$ that host all VNFs in F. To be specific, $loc_{i,j}^k$ is the ID of the node where $f_j \in F$ is to be placed in $Path(i)$. For example, $loc_{2,3}^k = 4$ means that VNF f_3 is to be placed on $node4$ along $Path(2)$. Besides, we show the structure of Y_k in Fig. 5, where $Path^k(i)$ denotes the path from s to t_i in solution Y_k.

In the fitness evaluation, for each solution, we first check its feasibility. A solution is said to be feasible if it satisfies Eq. (4) and inequalities (5) and (6); otherwise, it is said to be infeasible. For each infeasible solution, we set a sufficiently large number as its fitness value. For each feasible one, its fitness value is set to the total cost of the corresponding multicast tree, according to Eq. (10).

As mentioned above, within a multicast tree, some paths may overlap partially. However, there is no need to place two or

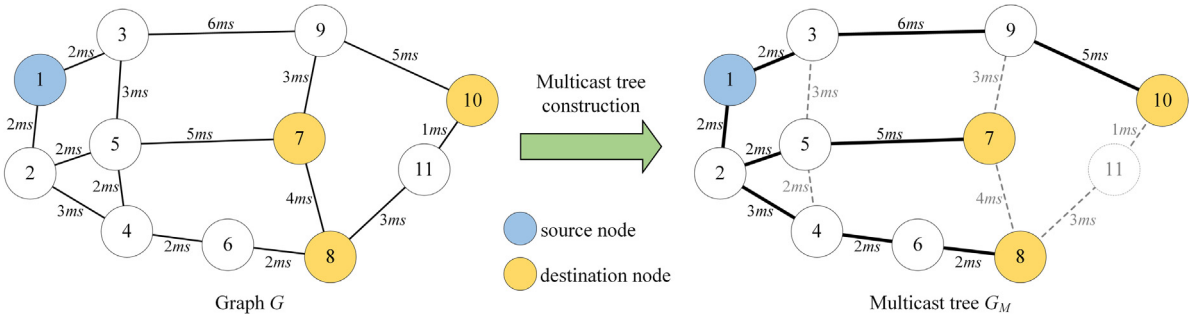

Fig. 4. An example network and its resultant multicast tree.

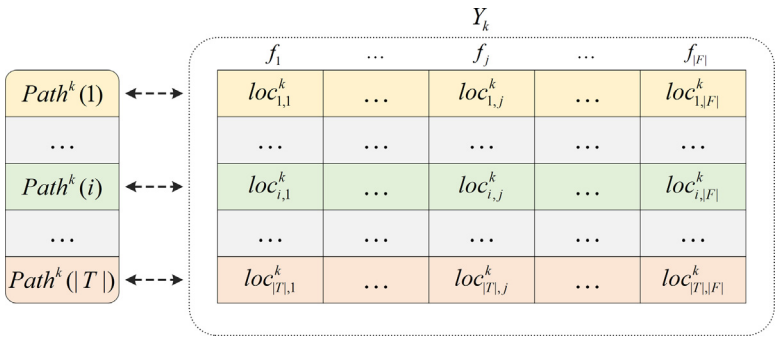

Fig. 5. Structure of solution Y_k.

more VNFs with the same functionality along the same overlapped section because the corresponding paths can share. In order to increase the utilization of each VNF, for any $Path(i)$, $i = 1, 2, \ldots, |T|$, we check if the following constraint is met, as defined in Eq. (13). If a solution does not satisfy the constraint, we penalize it by multiplying its fitness value by 1.5.

$$U(v_m) \cap U(v_n) = \varnothing, \forall v_m, v_n \in V^i, m \neq n \tag{13}$$

Fig. 6 depicts an example of solution representation. There are three paths in the multicast tree and three VNFs, f_1, f_2, f_3, in SFC. In solution Y, each row clearly reflects where each VNF is placed. For example, (A, C, C) in the first row means that f_1, f_2, and f_3 are hosted by nodes A, C, and C in $Path(1)$, respectively.

Different from the 1-dimensional encoding, 2DPSSE not only explicitly shows all locations for hosting VNFs along each path but also helps nEDA to explore the potential relationship among decision variables of the VNFP problem in multicast (see Section 4.2.2 for details). Besides, 2DPSSE helps significantly increase the probability of generating feasible solutions, making it appropriate for large-scale problems.

4.2.2. Solution generation based on three probability vectors

As aforementioned, the same set of VNFs is to be placed along each path within a constructed multicast tree. It is hence natural to place these VNFs path by path. In this paper, all paths are numbered according to destination ID. A path with a smaller destination ID is considered for VNFP earlier. For the very first path, i.e., $Path(1)$, one needs to choose carefully the location for hosting the very first VNF in the given SFC, because this location has a great impact on the quality of the entire VNFP. Besides, for an arbitrary path (except the first one), i.e., $Path(2), Path(3), \ldots, Path(|T|)$, the location where the very first VNF is placed, is also important. If it is not selected properly, the corresponding VNFP along that path does not have a sufficiently good point to begin with. Meanwhile, for partially overlapped paths, the more VNFs are placed along with the overlapped parts, the fewer compute nodes are needed for hosting them, consuming less compute resources. When considering the placement of all VNFs for a path, it is wise to make use of the locations of the VNFs already placed along other paths. This helps facilitate the sharing of VNFs among those paths that partially overlap. In addition, for each path within the multicast tree, we expect its associated data-flow to traverse each node only once. When considering the placement of one VNF for a path, it is important to consider the locations along the path that already host VNFs.

Fig. 7 shows an example of appropriate and inappropriate VNFP. There is a multicast tree with *node*1 as the source node, and *node*8 and *node*9 as two destination nodes. There are three VNFs to be placed, namely *VNF*1, *VNF*2, and *VNF*3. Each number in data-flow stands for the number of VNFs that the data-flow has already been processed. For example, "0" associated with data-flow 1 in Fig. 7(c) indicates the data-flow is not processed by VNF1 hosted on *node*2. In fact, the VNF1 hosted on *node*3 is used to process data-flow 1, as the number has changed to "1" after the data-flow passes by *node*3. Fig. 7(a) is an appropriate placement, where *node*2, *node*3, *node*4, and *node*5 are used to host VNFs. Fig. 7(b) shows an inappropriate placement due to the inappropriate location chosen for hosting *VNF*1. Compared with Fig. 7(a), (b) needs one more compute node. Fig. 7(c) is also an inappropriate placement, as *VNF*1 is placed twice. In Fig. 7(d), due to the improper ordering of the locations for hosting VNFs, the two data-flows are transmitted forward and backward multiple times, which leads to additional bandwidth consumption and larger end-to-end delay.

In this paper, there are three steps to generate a solution to the VNFP sub-problem. The first step determines where to place the very first VNF in the first path. The second step determines where to place the very first VNF in each path except the first one. The third step generates the locations for hosting the other VNFs in all paths.

We design three probability vectors to realize the three steps above. To be specific, the first probability vector (PV) is an initial placement location PV (IPL-PV). Given a constructed multicast

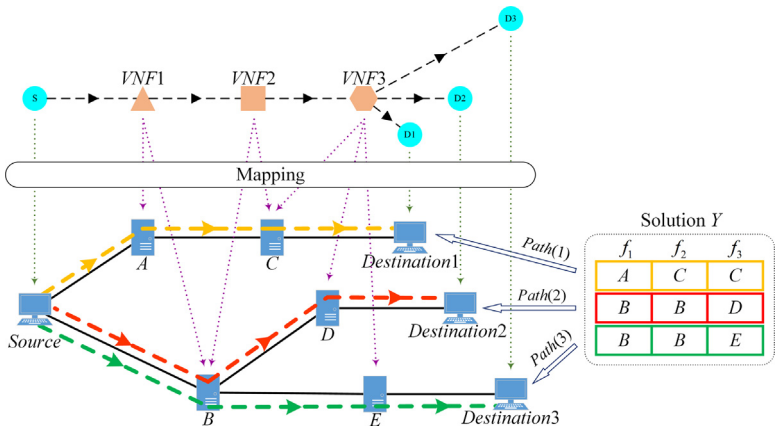

Fig. 6. An example of solution representation.

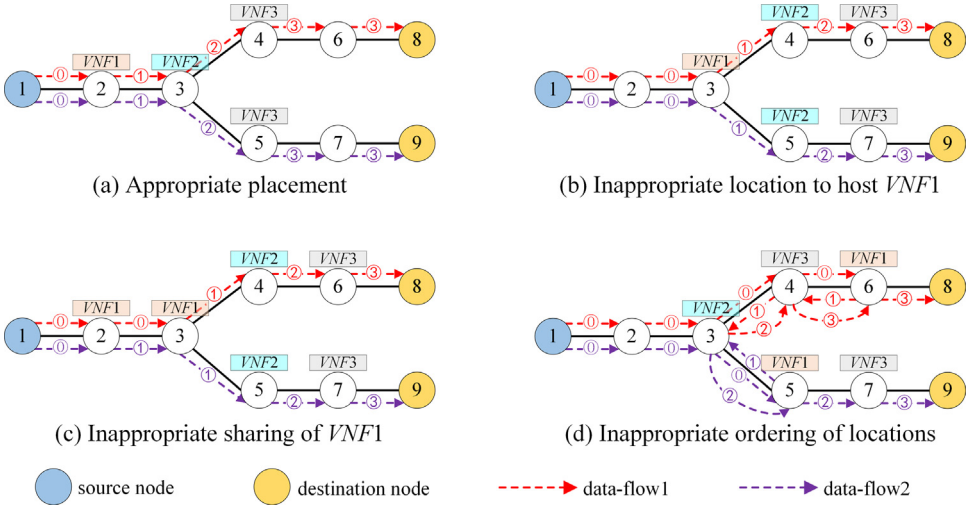

Fig. 7. An example of appropriate and inappropriate VNFP.

tree and its associated SFC, IPL-PV is responsible for generating the location where the very first VNF in the SFC is placed, in the very first path. The second one is an inter-path location dependency PV (ITPLD-PV), responsible for generating the locations for hosting the very first VNF in the other paths. ITPLD-PV generates locations path by path according to the last location already determined. By exploiting the dependency relationship between two adjacent locations, ITPLD-PV, to a certain extent, helps increase the probability of VNF sharing. The third one is an inner-path location dependency PV (INPLD-PV), which generates a location for hosting a particular VNF in each path, according to the location determined for hosting the precursor of this VNF. INPLD-PV takes the ordering of the locations for hosting VNFs along the same path into account, which helps decrease the probability of flow detouring.

Fig. 8 illustrates the relationship among the three proposed probability vectors. We generate solution Y by orderly sampling IPL-PV, ITPLD-PV, and INPLD-PV, once. Let $loc_{i,j}$ be the ID of the node where VNF $f_j \in F$ is to be placed in $Path(i)$, $i = 1, 2, \ldots, |T|, j = 1, 2, \ldots, |F|$. IPL-PV is used to generate location $loc_{1,1}$. ITPLD-PV is responsible for generating $|T| - 1$ locations, namely, $loc_{2,1}, \ldots, loc_{|T|-1,1}, loc_{|T|,1}$. Besides, INPLD-PV generates the locations for hosting the rest of the VNFs in each path.

(1) *IPL-PV*

IPL-PV is responsible for selecting a node in $Path(1)$ to host f_1. Denote IPL-PV by $\boldsymbol{P}^{IP}$. It is defined in Eq. (14).

$$\boldsymbol{P}^{IP} = \left[p_1^{IP}, p_2^{IP}, \ldots, p_{|V^1|}^{IP}\right] \tag{14}$$

where,

$$\sum_{k=1}^{|V^1|} p_k^{IP} = 1 \tag{15}$$

Element $p_k^{IP} \in [0, 1]$, $k = 1, 2, \ldots, |V^1|$, stands for the probability of selecting the kth node of V^1 to host f_1.

(2) *ITPLD-PV*

For an arbitrary path $Path(i)$, $i = 2, 3, \ldots, |T|$, ITPLD-PV is responsible for selecting a node in $Path(i)$ to host f_1. To be specific, location $loc_{i,1}$ is selected according to location $loc_{i-1,1}$ that has been already determined. We define ITPLD-PV, $\boldsymbol{P}^{IT}$, in Eq. (16).

$$\boldsymbol{P}^{IT} = \left[\boldsymbol{p}_2^{IT}, \boldsymbol{p}_3^{IT}, \ldots, \boldsymbol{p}_{|T|}^{IT}\right] \tag{16}$$

where,

$$\boldsymbol{p}_i^{IT} = \begin{bmatrix} \rho_{1,1}^{IT(i)} & \cdots & \rho_{1,|V^i|}^{IT(i)} \\ \vdots & \ddots & \vdots \\ \rho_{|V^{i-1}|,1}^{IT(i)} & \cdots & \rho_{|V^{i-1}|,|V^i|}^{IT(i)} \end{bmatrix}, i = 2, 3, \ldots, |T| \tag{17}$$

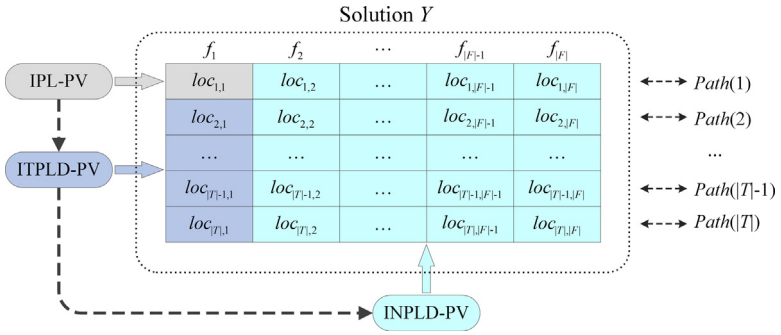

Fig. 8. Relationship among IPL-PV, ITPLD-PV, and INPLD-PV.

$$\sum_{n=1}^{|V^i|} \rho_{m,n}^{IT(i)} = 1, m = 1, 2, \ldots, |V^{i-1}| \tag{18}$$

where, $\boldsymbol{p}_i^{IT}$ is a probability matrix containing all probabilistic information about selecting location $loc_{i,1}$ to host f_1. $\rho_{m,n}^{IT(i)}$ is the probability of selecting the nth node in $Path(i)$ to host f_1, given that f_1 is placed on the mth node in $Path(i-1)$, $m = 1, 2, \ldots, |V^{i-1}|$, $n = 1, 2, \ldots, |V^i|$.

Let $\mathrm{p}\left(loc_{i,j}\right)$ denote the probability of selecting a certain node to host f_j in V^i, $i = 1, 2, \ldots, |T|, j = 1, 2, \ldots, |F|$. In fact, $\rho_{m,n}^{IT(i)}, i = 2, 3, \ldots, |T|$, is a conditional probability equal to $\mathrm{p}\left(loc_{i,1}|loc_{i-1,1}\right)$, where $loc_{i,1}$ and $loc_{i-1,1}$ are the nth node in $Path(i)$ and the mth node in $Path(i-1)$, respectively. The probability distribution of $\mathrm{p}\left(loc_{1,1}, loc_{2,1}, \ldots, loc_{|T|,1}\right)$ is decomposed by Eq. (19).

$$\begin{aligned}&\mathrm{p}\left(loc_{1,1}, loc_{2,1}, \ldots, loc_{|T|,1}\right)\\ &\quad = \mathrm{p}\left(loc_{1,1}\right)\mathrm{p}\left(loc_{2,1}|loc_{1,1}\right)\ldots\mathrm{p}\left(loc_{|T|,1}|loc_{|T|-1,1}\right)\end{aligned} \tag{19}$$

where, $\mathrm{p}\left(loc_{i,1}|loc_{i-1,1}\right)$ is the conditional probability of $loc_{i,1}$ given $loc_{i-1,1}$, $i = 2, 3, \ldots, |T|$. In other words, ITPLD-PV is, to a certain extent, able to exploit the pairwise dependency relationship between two locations in two adjacent paths, as shown in Fig. 9.

(3) *INPLD-PV*

For an arbitrary path $Path(i)$, $i = 1, 2, \ldots, |T|$, INPLD-PV is responsible for selecting $|F| - 1$ nodes in $Path(i)$ for hosting VNFs, $f_2, f_3, \ldots, f_{|F|}$. To be specific, location $loc_{i,j}$ is selected according to location $loc_{i,j-1}$ that has been already determined, $j = 2, 3, \ldots, |F|$. We define INPLD-PV, $\boldsymbol{P}^{IN}$, in Eq. (20).

$$\boldsymbol{P}^{IN} = \begin{bmatrix} \boldsymbol{p}_{1,2}^{IN} & \cdots & \boldsymbol{p}_{1,|F|}^{IN} \\ \vdots & \ddots & \vdots \\ \boldsymbol{p}_{|T|,2}^{IN} & \cdots & \boldsymbol{p}_{|T|,|F|}^{IN} \end{bmatrix} \tag{20}$$

where,

$$\boldsymbol{p}_{i,j}^{IN} = \begin{bmatrix} \rho_{1,1}^{IN(i,j)} & \cdots & \rho_{1,|V^i|}^{IN(i,j)} \\ \vdots & \ddots & \vdots \\ \rho_{|V^i|,1}^{IN(i,j)} & \cdots & \rho_{|V^i|,|V^i|}^{IN(i,j)} \end{bmatrix},$$

$$i = 1, 2, \ldots, |T|, j = 2, 3, \ldots, |F| \tag{21}$$

$$\sum_{n=1}^{|V^i|} \rho_{m,n}^{IN(i,j)} = 1, m = 1, 2, \ldots, |V^i| \tag{22}$$

where, $\boldsymbol{p}_{i,j}^{IN}$ is a probability matrix containing all probabilistic information about selecting all locations along $Path(i)$ for hosting $f_2, f_3, \ldots, f_{|F|}$. $\rho_{m,n}^{IN(i,j)}$ stands for the probability of selecting the nth node in $Path(i)$ to host f_j, given that f_{j-1} is placed on the mth node in $Path(i)$, $m, n = 1, 2, \ldots, |V^i|$. Similarly, the probability distribution of $\mathrm{p}\left(loc_{i,1}, loc_{i,2}, \ldots, loc_{i,|F|}\right)$ is decomposed by Eq. (23).

$$\begin{aligned}&\mathrm{p}\left(loc_{i,1}, loc_{i,2}, \ldots, loc_{i,|F|}\right)\\ &\quad = \mathrm{p}\left(loc_{i,1}\right)\mathrm{p}\left(loc_{i,2}|loc_{i,1}\right)\ldots\mathrm{p}\left(loc_{i,|F|}|loc_{i,|F|-1}\right)\end{aligned} \tag{23}$$

where, $\mathrm{p}\left(loc_{i,j}|loc_{i,j-1}\right)$ is the conditional probability of $loc_{i,j}$ given $loc_{i,j-1}$, $i = 1, 2, \ldots, |T|, j = 2, 3, \ldots, |F|$. INPLD-PV determines locations for hosting VNFs based on chain-like pairwise dependency relationship between every two adjacent locations in each path, as shown in Fig. 10.

4.2.3. The flexible probability vector update scheme

In EDA, a PV generates a set of solutions in each iteration to explore multiple areas of interest in the search space. Updating the PV is, no doubt, the main evolutionary force that helps guide the search towards promising areas where high-quality solutions may reside. If a PV is not updated properly, the search is easily trapped into local optima due to the rapid loss in population diversity. In the literature, however, a considerable amount of EDAs suffers from premature convergence since their PV update schemes cannot keep an appropriate level of diversity during the evolution [52]. This paper proposes a flexible PV update (FPVU) scheme, where elitist solutions are utilized to guide the search towards global optima, and a repair method is devised to modify probabilistic distributions for enhancing diversification.

The FPVU scheme maintains a set of elitist solutions, *ES*, consisting of N_{ES} best-so-far solutions found during the evolution, where N_{ES} is a positive integer no larger than population size N_{pop}. In each iteration, the statistical data obtained from *ES* is used to update the three PVs, including IPL-PV, ITPLD-PV, and INPLD-PV. Then, for each PV, an element equal to 0 is set to a value larger than 0, which helps avoid local optima.

At the beginning of the evolution, the N_{ES} best solutions in the initial population are copied into *ES*. Then, *ES* is updated by the best-so-far solutions obtained in each iteration. In terms of the PV update, we use three weight sets to update the three PVs in Section 4.2.2.

The first weight set, $\boldsymbol{W}^{IP}$, is used to update IPL-PV, as defined in Eq. (24).

$$\boldsymbol{W}^{IP} = \left\{w_1^{IP}, w_2^{IP}, \ldots, w_{|V^i|}^{IP}\right\} \tag{24}$$

where w_k^{IP}, $k = 1, 2, \ldots, |V^1|$, counts how many times the kth node in $Path(1)$ is selected to host f_1, based on all solutions in *ES*.

Fig. 11 shows the four solutions corresponding to the four cases in Fig. 7. To be specific, Y_a, Y_b, Y_c and Y_d are the solutions to Fig. 7(a), (b), (c) and (d), respectively. Obviously, in Y_a, *node*2 in $Path(1)$ is selected to host f_1; in Y_b and Y_c, *node*3 in $Path(1)$ is selected to host f_1; in Y_d, *node*6 in $Path(1)$ is selected to host f_1. Hence, we have $w_1^{IP} = 1$, $w_2^{IP} = 2$, $w_3^{IP} = 0$ and $w_4^{IP} = 1$.

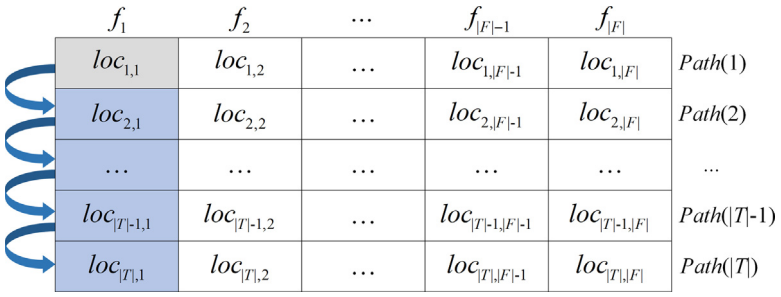

Fig. 9. Pairwise dependency between two locations in two adjacent paths.

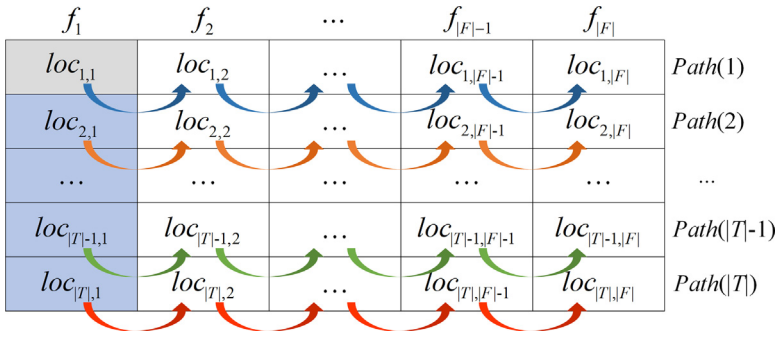

Fig. 10. Pairwise dependency between every two adjacent locations in each path.

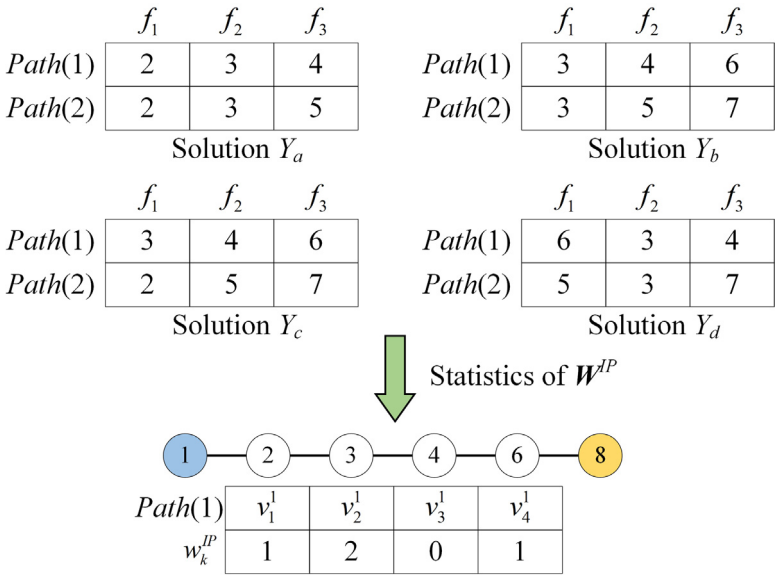

Fig. 11. Four solutions and the statistics of $\boldsymbol{W}^{IP}$.

We update each element in $\boldsymbol{P}^{IP}$ by Eq. (25).

$$p_k^{IP} = \frac{w_k^{IP}}{\sum_{q=1}^{|V^1|} w_q^{IP}}, \forall k \in \{1, 2, \ldots, |V^1|\} \tag{25}$$

Note that after $\boldsymbol{P}^{IP}$ is updated, some elements in $\boldsymbol{P}^{IP}$ may be 0, which means the corresponding nodes in $Path(1)$ have never been selected for hosting f_1 in the solutions in ES. However, this does not necessarily mean that hosting f_1 on them cannot result in promising or optimal solutions. Rather, it could lead to local optima due to the bias of best so far solutions. In order to enhance the global exploration ability of the search, it is necessary to address the case of $p_k^{IP} = 0$, for $k \in \{1, 2, \ldots, |V^1|\}$. We thus devise a zero-repair method to modify $\boldsymbol{P}^{IP}$ so that each element previously equal to 0 is set to a small probability. Each element, p_k^{IP}, $k = 1, 2, \ldots, |V^1|$, is updated by Eq. (26).

$$p_k^{IP} = \begin{cases} \dfrac{\beta}{N_{zeros}}, & \text{if } p_k^{IP} = 0 \\ (1-\beta) \cdot p_k^{IP}, & \text{otherwise} \end{cases} \tag{26}$$

where, β $(0 < \beta < 1)$ is a repair parameter controlling how severe the probabilistic distribution is shifted to each element with a value of 0. A smaller β results in a less severe change to the previous probabilistic distribution. N_{zeros} is the number of elements in $\boldsymbol{P}^{IP}$ that are equal to 0 after the calculation of Eq. (25).

The second weight set, $\boldsymbol{W}^{IT}$, is used to update ITPLD-PV, as defined in Eq. (27).

$$\boldsymbol{W}^{IT} = \left\{ \boldsymbol{w}^{IT(i,m)} | i = 2, 3, \ldots, |T| ; m = 1, 2, \ldots, |V^{i-1}| \right\} \tag{27}$$

where, $\boldsymbol{w}^{IT(i,m)}$ is defined in Eq. (28).

$$\boldsymbol{w}^{IT(i,m)} = \left\{ \omega_1^{IT(i,m)}, \omega_2^{IT(i,m)}, \ldots, \omega_{|V^i|}^{IT(i,m)} \right\} \tag{28}$$

where, $\omega_n^{IT(i,m)}$, $n = 1, 2, \ldots, |V^i|$, counts how many times the nth node in $Path(i)$ is selected to host f_1, given that f_1 is placed on the mth node in $Path(i-1)$, based on all solutions in ES.

We update each element in $\boldsymbol{P}^{IT}$, $\rho_{m,n}^{IT(i)}$, $i = 2, 3, \ldots, |T|$, $m = 1, 2, \ldots, |V^{i-1}|$, $n = 1, 2, \ldots, |V^i|$, by Eq. (29). After all elements are updated, the zero-repair method is also adopted.

$$\rho_{m,n}^{IT(i)} = \frac{\omega_n^{IT(i,m)}}{\sum_{q=1}^{|V^i|} \omega_q^{IT(i,m)}} \tag{29}$$

The third weight set, $\boldsymbol{W}^{IN}$, is used to update INPLD-PV, as defined in Eq. (30).

$$\boldsymbol{W}^{IN} = \left\{ \boldsymbol{w}^{IN(i,j,m)} | i = 1, 2, \ldots, |T| ; j = 2, 3, \ldots, |F| ; m = 1, 2, \ldots, |V^i| \right\} \tag{30}$$

where, $\boldsymbol{w}^{IN(i,j,m)}$ is defined in Eq. (31).

$$\boldsymbol{w}^{IN(i,j,m)} = \left\{ \omega_1^{IN(i,j,m)}, \omega_2^{IN(i,j,m)}, \ldots, \omega_{|V^i|}^{IN(i,j,m)} \right\} \tag{31}$$

where, $\omega_n^{IN(i,j,m)}$, $n = 1, 2, \ldots, |V^i|$, counts how many times the nth node in $Path(i)$ is selected to host f_j, given that f_{j-1} is placed on the mth node in $Path(i)$, based on all solutions in ES.

We update each element in $\boldsymbol{P}^{IN}$, $\rho_{m,n}^{IN(i,j)}$, $i = 1, 2, \ldots, |T|$, $j = 2, 3, \ldots, |F|$, $m, n = 1, 2, \ldots, |V^i|$, by Eq. (32). Again, after all elements are updated, the zero-repair method is applied.

$$\rho_{m,n}^{IN(i,j)} = \frac{\omega_n^{IN(i,j,m)}}{\sum_{q=1}^{|V^i|} \omega_q^{IN(i,j,m)}} \tag{32}$$

The FPVU scheme makes use of the best-so-far solutions obtained during the evolution to drive the search towards high-quality solutions and modifies probabilistic distributions of the three PVs to avoid zero-probability cases when selecting locations, which helps enhance global exploration and avoid prematurity.

4.2.4. Overall procedure of nEDA

nEDA is featured with the 2DPSSE scheme in Section 4.2.1, the three PVs in Section 4.2.2, as well as the FPVU scheme in Section 4.2.3. The overall procedure of nEDA is given in Algorithm 2. The search stops once a predefined number of iterations are run.

PV mutation is one of the commonly used methods for diversity preservation, where a small disturbance is introduced to a PV in each iteration [53]. This paper applies simple mutation [51] to IPL-PV, ITPLD-PV, and INPLD-PV to further maintain diversity level during the evolution.

4.3. Complexity analysis

This paper proposes a two-stage approach to solve the MVNFP problem. In the first stage, Dijkstra's algorithm [45] is adopted to construct a multicast tree consisting of $|T|$ paths. To run this algorithm once, we can obtain a path connecting the source and a destination. Dijkstra's algorithm has a time complexity of $O(|V|^2)$. Hence, finding $|T|$ paths requires a time complexity of $O(|V|^2 \cdot |T|)$.

In the second stage, nEDA is used to place VNFs over the multicast tree constructed. The procedure of nEDA includes the initialization and the main loop. Let $O(\mathcal{F})$ be the time complexity for evaluating a solution. Let $Path(max)$ be the longest path among all paths, where $|V^{max}| = \max\left\{|V^i| | i = 1, 2, \ldots, |T|\right\}$ is the number of nodes in $Path(max)$. In the initialization, generating N_{pop} solutions and evaluating them in Step 1 results in a time complexity of $O(\mathcal{F} \cdot N_{pop})$. In Step 4, the time complexities of IPL-PV, ITPLD-PV and INPLD-PV initialization are $O(|V^1|)$, $O(|T| \cdot |V^{max}|^2)$ and $O(|T| \cdot |F| \cdot |V^{max}|^2)$, respectively. Actually, compared with the fitness evaluation, Steps 2–4 are trivial and can be ignored. So, the initialization has a time complexity of $O(\mathcal{F} \cdot N_{pop})$.

The main loop consists of population reproduction (Steps 6–10), fitness evaluation (Step 11), update of elitist solutions (Step 12), the FPVU scheme (Step 13), and simple mutation (Step 14). In the population reproduction, to generate a solution is to sample IPL-PV, ITPLD-PV and INPLD-PV, once, which leads to a time complexity of $O\left(|V^1| + (|T|-1) \cdot |T| \cdot |V^{max}|^2 + (|T|-1) \cdot (|F|-1) \cdot |T| \cdot |F| \cdot |V^{max}|^2\right) = O\left(|T|^2 \cdot |F|^2 \cdot |V^{max}|^2\right)$. Thus, to generate N_{pop} solutions requires a time complexity of $O\left(N_{pop} \cdot |T|^2 \cdot |F|^2 \cdot |V^{max}|^2\right)$. In the fitness evaluation, all solutions are evaluated, which results in a time complexity of $O\left(\mathcal{F} \cdot N_{pop}\right)$. To update ES, we sort the population by their fitness values to obtain N_{ES} best solutions. The time complexity of sorting the N_{pop} solutions is $O\left(N_{pop} \cdot \log N_{pop}\right)$ and that of selecting the N_{ES} best solutions is $O\left(N_{ES}\right)$. As $N_{ES} \ll N_{pop}$ in this paper, updating ES has a time complexity of $O\left(N_{pop} \cdot \log N_{pop} + N_{ES}\right) = O\left(N_{pop} \cdot \log N_{pop}\right)$. In the FPVU scheme, IPL-PV, ITPLD-PV, and INPLD-PV are updated by three weight sets that are generated based on the statistics of the N_{ES} best solutions, respectively. As each solution is a $|T| \times |F|$ vector, the time complexity of generating the three weight sets is $O\left(N_{ES} \cdot |T| \cdot |F|\right)$. To update the three PVs requires time complexities of $O\left(|V^1|\right)$, $O\left(|T| \cdot |V^{max}|^2\right)$, and $O\left(|T| \cdot |F| \cdot |V^{max}|^2\right)$, respectively. Hence, the time complexity of the FPVU scheme can be written as $O\left(N_{ES} \cdot |T| \cdot |F| + |V^1| + |T| \cdot |V^{max}|^2 + |T| \cdot |F| \cdot |V^{max}|^2\right) = O(N_{ES} \cdot |T| \cdot |F| + |T| \cdot |F| \cdot |V^{max}|^2)$. Besides, the simple mutation has a time complexity of $O(N_{pop} \cdot |T| \cdot |F|)$. Hence, the overall time complexity of the second stage is $O(N_{pop} \cdot |T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F} \cdot N_{pop} + N_{pop} \cdot \log N_{pop} + N_{ES} \cdot |T| \cdot |F| + |T| \cdot |F| \cdot |V^{max}|^2 + N_{pop} \cdot |T| \cdot |F|) = O(N_{pop} \cdot |T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F} \cdot N_{pop} + N_{pop} \cdot \log N_{pop})$. Due to the MVNFP problem is highly constrained, the fitness evaluation has a much higher time complexity than the ES update process. Thus, the time complexity of the main loop is reduced to $O(N_{pop} \cdot |T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F} \cdot N_{pop})$. Therefore, the overall time complexity of the second stage is $O\left(\mathcal{F} \cdot N_{pop} + N_{iter} \cdot \left(N_{pop} \cdot |T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F} \cdot N_{pop}\right)\right) = O\left(N_{iter} \cdot N_{pop} \cdot \left(|T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F}\right)\right)$.

Compared with the second stage, the time complexity of the first stage is too trivial to take into account. So, the total time complexity of the two-stage approach is equal to $O\left(N_{iter} \cdot N_{pop} \cdot \left(|T|^2 \cdot |F|^2 \cdot |V^{max}|^2 + \mathcal{F}\right)\right)$.

5. Performance evaluation

In this section, we first introduce the test instances and parameter settings. Then, we evaluate the performance of nEDA in the context of VNFP in multicast. After that, we evaluate the overall performance of the proposed two-stage approach for the MVNFP problem (TS-M).

5.1. Test instances

As the MVNFP problem concerned in this paper has not attracted enough research attention, no benchmark instance is immediately available. We generate 18 test instances for performance evaluation, including six real-world networks and 12 ran-

Algorithm 2 nEDA for VNF placement

Input: a multicast service request MSR, a multicast tree G_M constructed by Algorithm 1, and other related parameters α, β, N_{pop}, N_{iter} and N_{ES}.

Output: The best solution found Y_{best}.

// **Initialization:**

1. Randomly generate an initial population of N_{pop} solutions and evaluate them;
2. Select the N_{ES} best solutions from the population;
3. Set $ES = \emptyset$ and place the N_{ES} best solutions into ES;
4. Initialize IPL-PV, ITPLD-PV and INPLD-PV by ES; // see Subsection 4.2.3

// **Repeat:**

5. **for** $m = 1$ **to** N_{iter} **do** // in each iteration
6. **for** $k = 1$ **to** N_{pop} **do** // see Subsection 4.2.2
7. Generate a location in $Path(1)$ to host f_1 for Y_k by sampling $\boldsymbol{P}^{IP}$;
8. **for** $i = 2$ **to** $|T|$ **do**
9. Generate a location in $Path(i)$ to host f_1 for Y_k by sampling $\boldsymbol{P}^{IT}$;
   for $i = 1$ **to** $|T|$ **do**
   for $j = 2$ **to** $|F|$ **do**
10. Generate all locations along $Path(i)$, to host f_j for Y_k by sampling $\boldsymbol{P}^{IN}$;
11. Evaluate the N_{pop} generated solutions; // see Subsection 4.2.1
12. Update ES by finding the N_{ES} best solutions from the current population and ES;
13. Update $\boldsymbol{P}^{IP}$, $\boldsymbol{P}^{IT}$ and $\boldsymbol{P}^{IN}$ by the FPVU scheme; // see Subsection 4.2.3
14. Apply simple mutation in [51] to $\boldsymbol{P}^{IP}$, $\boldsymbol{P}^{IT}$ and $\boldsymbol{P}^{IN}$;

Table 3
Test instances and their parameters.

Instance ID	Instance name	Nodes	Links	Destinations	VNFs in SFC
I-1	Dfn	58	87	8	4
I-2	Germany50	50	88	7	6
I-3	Kentucky	754	895	9	5
I-4	Sun	27	102	4	4
I-5	Tata	145	186	5	6
I-6	Tinet	53	89	5	6
I-7	N50_1	50	123	7	5
I-8	N50_2	50	128	6	4
I-9	N100_1	100	463	8	5
I-10	N100_2	100	405	7	4
I-11	N150_1	150	466	7	5
I-12	N150_2	150	419	7	4
I-13	N200_1	200	444	9	4
I-14	N200_2	200	456	9	5
I-15	N250_1	250	727	9	5
I-16	N250_2	250	665	9	4
I-17	N300_1	300	987	9	4
I-18	N300_2	300	990	9	4

domly generated networks. Among the real-world networks, Germany50 and Sun are from SDNlib [54], and Dfn, Kentucky, Tata, and Tinet are from the Internet Topology Zoo [55]. The rest of the instances are randomly generated by the random ER graph generation algorithm [56]. Table 3 shows the test instances and their parameters. To encourage scientific comparison in the future, we make the 18 test instances available at: http://userweb.swjtu.edu.cn/Userweb/hxx/research.htm.

In each test instance, the multicast service request is randomly generated. To be specific, the source, the set of destinations, the set of ordered VNFs, as well as the set of bandwidth requirements are randomly generated. For each node v, its available compute resource R_v^{avlb} is uniformly distributed in the range of [30, 50] *units*. For each VNF $f_j \in F$, its required compute resource $R_{f_j}^{csmd}$ is uniformly distributed in the range of [10, 20] *units*. The processing delay incurred by f_j in $Path(i)$, $D_{prcs}(v_k^i, f_j)$, is randomly generated in the range of [1, 10] ms. For each link $e \in E$, its associated propagation delay, $D_{prpg}(e)$, is calculated based on the length of e. Besides, the available bandwidth, $B(e)$, is uniformly distributed in the range of [100, 150] Mbps. In addition, the bandwidth requirement between f_j and f_{j+1}, $bw_{j,j+1}$, is randomly generated in the range of [10, 30] Mbps.

5.2. Performance evaluation of nEDA

To verify the performance of nEDA, we compare it with five state-of-the-art EAs, as listed in Table 4. To make a fair comparison, for each algorithm, we set the population size $N_{pop} = 100$ [22,24] and the predefined number of iterations $N_{iter} = 300$ [57,58], which are the two commonly parameter settings in EDAs. For nEDA, we set the size of ES, $N_{ES} = N_{pop}/5$ and the repair parameter, $\beta = 0.1$. For other EAs, we directly adopt their parameter settings [51,59–61] [62]. For IEPBIL, the learning rate is set to 0.01, and the mutation probability and mutation shift are set to 0.02 and 0.02, respectively. For RA-GA, we set the crossover and mutation probabilities to 0.7 and 0.01, respectively. For AP-ACO, the pheromone evaporation factor and the constant parameter are set to 0.7 and 30, respectively. Besides, the two heuristic factors are initialized as 0.1 and 5, respectively. For CGA, the crossover and mutation probabilities are set to 0.8 and 0.25, respectively. The top 20% of the best solutions of the current iteration are considered as the normative knowledge and the best-so-far solution is regarded as the situational knowledge. Note that for each instance, all algorithms begin with an identical multicast tree constructed by Dijkstra's algorithm.

As known, meta-heuristics are usually stochastic search algorithms. Running an algorithm multiple times probably leads to different objective function values. Running a stochastic search algorithm 20 times is meaningful from the point of view of statistics, i.e., the results collected to some extent reflect that algorithm's optimization performance [63,64]. All results are collected by running each algorithm 20 times [65,66] on a machine with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz and 16 GB RAM.

Fig. 12 illustrates the best fitness curves obtained by the six algorithms in 18 test instances. nEDA performs the best as it

Table 4
Description of the five state-of-the-art EAs.

Algorithm	Author	Description
IEPBIL	Xing et al. [51]	The integer-encoding population based incremental learning algorithm that tackles the unicast-oriented VNFP problem with delay constraint.
RA-GA	Carpio et al. [59]	The resource allocation genetic algorithm for unicast-oriented VNFP, where load balancing is taken into account.
AP-ACO	Wei et al. [60]	The ant colony optimization based on adaptive parameter setting. AP-ACO is proposed to address the resource utilization problem in virtual machine (VM) placement. Note that VM placement considers how VMs are economically placed on physical machines, which is similar to the unicast-oriented VNFP problem.
SSA	Abualigah et al. [61]	The salp swarm algorithm mimicking the swarming behavior of salps in oceans. SSA has been successfully applied in a variety of areas, such as feature selection [67], engineering optimization [68] and VM placement [69].
CGA	Laaziz et al. [62]	The cultural genetic algorithm proposed to handle medium and large-scale unicast-oriented VNFP problem, where the best candidate solutions are regarded as knowledge.

Table 5
Results of ABF values (Best results are in bold).

Algorithm	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9
nEDA	**93.69**	**204.17**	**600.41**	**339.19**	**223.23**	**310.76**	**340.17**	**152.06**	**417.46**
IEPBIL	163.13	660.29	933.19	345.43	500.01	850.70	834.10	257.78	691.38
RA-GA	217.65	651.11	946.52	408.33	489.56	808.42	851.33	321.36	791.43
AP-ACO	276.78	571.16	968.07	605.81	334.47	1167.37	1145.53	251.74	856.66
SSA	349.42	1167.73	1000.69	680.87	628.85	1082.57	1162.71	381.60	905.02
CGA	296.76	969.26	957.53	655.98	516.53	1054.91	890.56	344.19	815.76
Algorithm	I-10	I-11	I-12	I-13	I-14	I-15	I-16	I-17	I-18
nEDA	**373.93**	**856.71**	**417.50**	**715.13**	**755.41**	**623.64**	**488.40**	**568.52**	**388.45**
IEPBIL	475.24	1355.24	840.58	1157.44	1199.42	1083.18	883.41	811.96	624.99
RA-GA	503.49	1366.77	872.99	1162.66	1239.72	1057.82	894.99	853.84	673.25
AP-ACO	575.00	1682.77	1317.74	1572.22	1554.46	1423.79	1243.41	1010.32	732.98
SSA	649.21	1639.66	1391.91	1634.50	1319.67	1384.01	1228.63	982.02	760.37
CGA	641.22	1544.83	959.68	1244.48	1373.03	1173.87	899.48	891.38	686.85

consistently achieves the best convergence curve in each test instance. Obviously, nEDA converges rapidly yet without premature evolution. This means it is not easily stuck into local optima, especially in the early stage of evolution, which helps to strike a balance between global exploration and local exploitation. On the one hand, the proposed FPVU scheme maintains a diversified population, helping to enhance global exploration. On the other hand, the three problem-specific PVs mine the inner relationship between each pair of paths, helping to improve the local exploitation. This is why nEDA achieves the best performance among the six EAs.

IEPBIL, as one of EDAs, is the second-best algorithm since it outperforms RA-GA, AP-ACO, SSA, and CGA in fourteen instances except I-2, I-5, I-8, and I-15. Nevertheless, IEPBIL is featured with premature convergence in most instances. This is because IEPBIL heavily relies on the best-so-far solution found during the evolution. If this solution remains the same in a relatively long period of evolution, it is extremely difficult for IEPBIL to escape from local optima. Unfortunately, this happens when addressing the MVNFP problem.

RA-GA sometimes gains decent convergence performance. It is, however, easily trapped into local optima in the early stage of evolution. The reason behind this is the commonly used crossover and mutation operations cannot produce a sufficient number of diversified solutions, which does not help carry out efficient exploration over multiple areas of interest in the search space. Since the MVNFP problem is highly constrained, RA-GA could not achieve expected performance unless problem-specific reproduction operations are developed.

CGA was developed to handle medium and large-scale unicast-oriented VNFP problems. However, it does not gain good performance with respect to the MVNFP problem, even in small-scale instances (see I-1 and I-9 in Fig. 12). The following explains why. CGA reproduces solutions by belief space, where knowledge simply relies on the best solutions. But, these solutions cannot provide sufficient evolutionary force to drive the exploration over vast areas in the search space, due to that MVNFP problems are much more complicated than unicast-oriented VNFP problems. In particular, the swapping mutation operation cannot maintain an appropriate level of diversity, which easily leads to prematurity.

AP-ACO and SSA are the two worst algorithms. On the one hand, according to Refs. [60,70] and [71], ACO is a useful tool to tackle both VM and VNFP problems. Nevertheless, AP-ACO does not perform well on the MVNFP problem. This is because the positive feedback mechanism in AP-ACO relies too much on the best-so-far solution. Although adaptive parameter tuning is adopted, AP-ACO still suffers from prematurity in most instances, leading to local optima. On the other hand, in SSA, the swarming behavior of salps is similar to a chain. SSA should achieve good performance since MVNFP considers the deployment of multiple VNF chains. Unfortunately, SSA performs the worst in all selected test instances. This is because a salp chain is a single chain with one leader and a number of followers. It cannot mimic multiple paths at the same time, given a multicast tree. Hence, SSA is not able to achieve decent performance in terms of global exploration. As the level of diversity is kept low, it makes sense that SSA cannot effectively address the problem concerned in this paper.

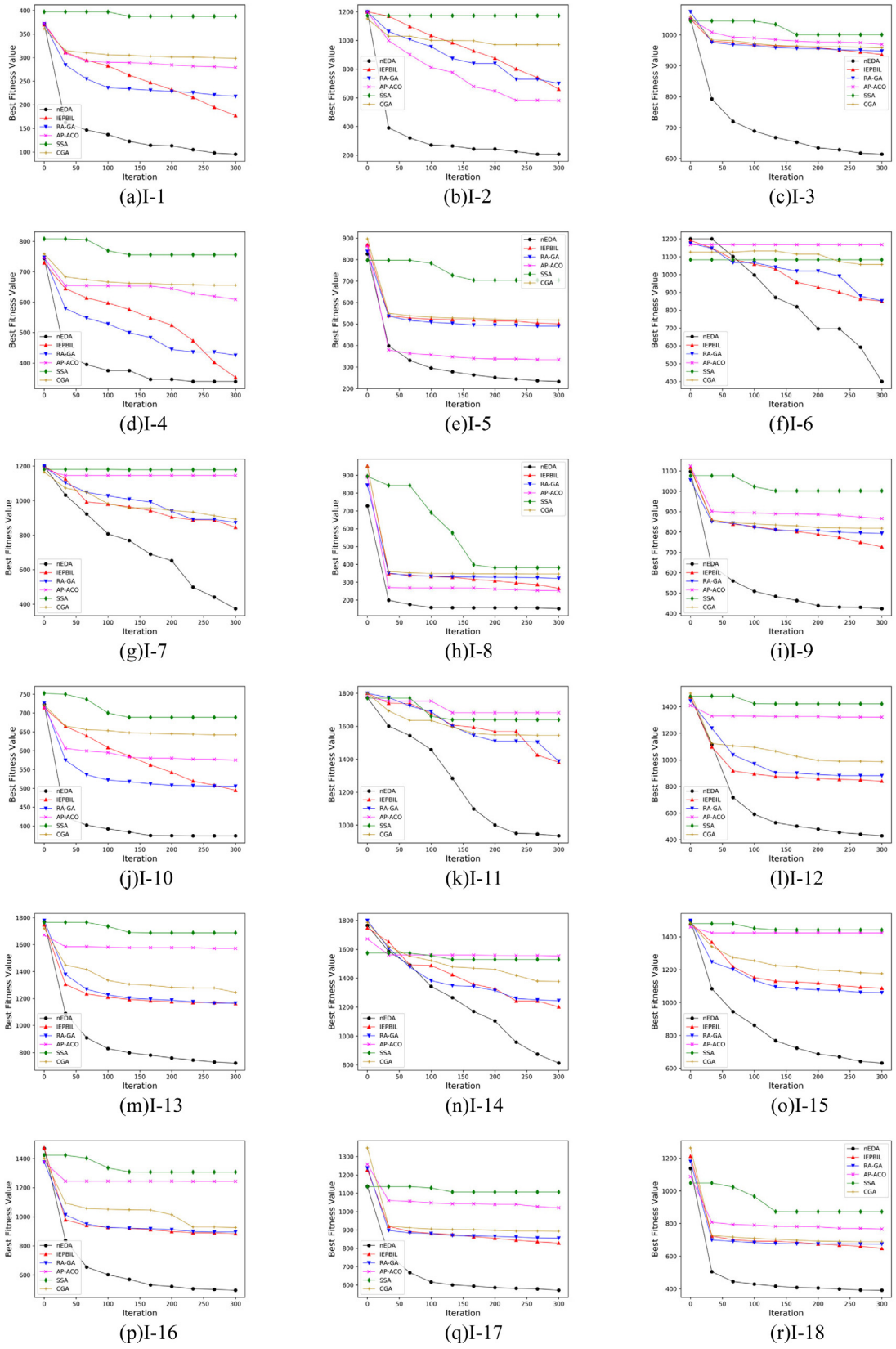

Fig. 12. Best fitness curves in 18 test instances.

Then, we compare the average best fitness (ABF) values obtained by the six EAs, as shown in Table 5. It is easily seen that nEDA is the best as it always obtains the smallest ABF values in each test instance. To support the observation above, we show the box-plots of the ABF values on 18 test instances in Fig. 13. Clearly, nEDA outperforms the other five EAs since its associated box is always at the bottom.

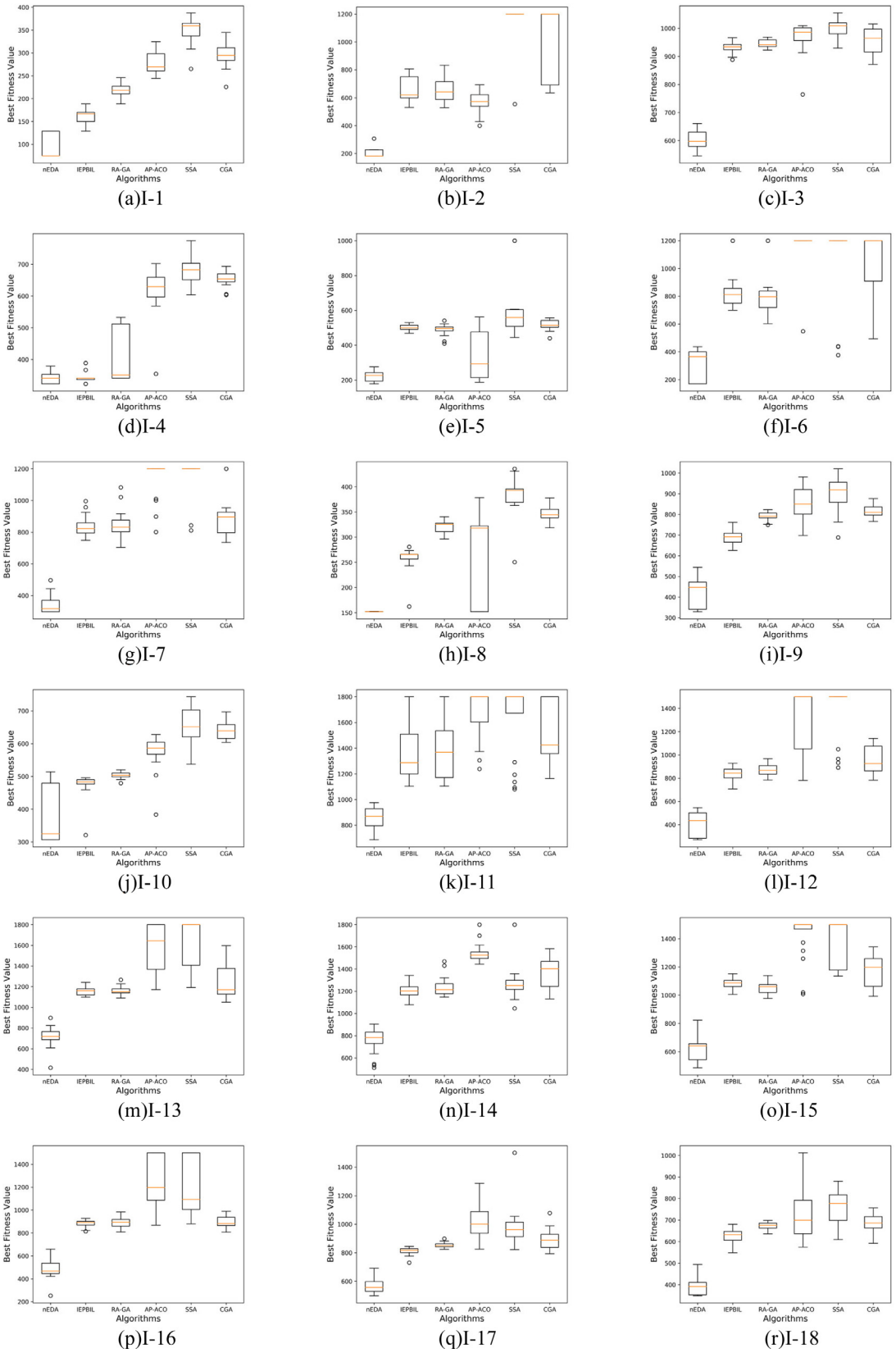

Fig. 13. Box-plots of the ABF values on 18 test instances.

Besides, we compare the six EAs using the Student's t-test and show the statistical results in Table 6. Two-tailed t-test with 38 degrees of freedom at a significance level of 0.05 is adopted [63]. Symbol "+" indicates Alg.1 is significantly better than Alg.2, while "$\sim$" means Alg.1 is statistically equal to Alg.2. It is no doubt that nEDA performs better than IEPBIL, RA-GA, AP-ACO, SSA, and CGA if considering all test instances. To be specific, nEDA outperforms IEPBIL in 17 instances except I-4, while it beats RA-GA, AP-ACO,

Table 6
Results of t-test.

Alg.1 ↔ Alg.2	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9
nEDA ↔ IEPBIL	+	+	+	∼	+	+	+	+	+
nEDA ↔ RA-GA	+	+	+	+	+	+	+	+	+
nEDA ↔ AP-ACO	+	+	+	+	+	+	+	+	+
nEDA ↔ SSA	+	+	+	+	+	+	+	+	+
nEDA ↔ CGA	+	+	+	+	+	+	+	+	+
Alg.1 ↔ Alg.2	I-10	I-11	I-12	I-13	I-14	I-15	I-16	I-17	I-18
nEDA ↔ IEPBIL	+	+	+	+	+	+	+	+	+
nEDA ↔ RA-GA	+	+	+	+	+	+	+	+	+
nEDA ↔ AP-ACO	+	+	+	+	+	+	+	+	+
nEDA ↔ SSA	+	+	+	+	+	+	+	+	+
nEDA ↔ CGA	+	+	+	+	+	+	+	+	+

Table 7
Rankings of six evolutionary algorithms.

Algorithm	nEDA	IEPBIL	RA-GA	AP-ACO	SSA	CGA
Average rank	1	2.39	2.94	4.61	5.61	4.44
Position	1	2	3	5	6	4

SSA, and CGA in all instances. In addition, the Friedman test [64] is also used for algorithm performance comparison and the average rankings of the six algorithms are shown in Table 7. With the convergence curve, ABF value, box-plot, t-test, and Friedman test, taken into consideration, one can observe that nEDA achieves the best optimization results. This indicates the proposed algorithm is more suitable for tackling the MVNFP problem studied in this paper than the existing state-of-the-art EAs.

The average computational time (ACT) is another commonly used performance indicator. Table 8 presents the ACT values of the six EAs. nEDA and IEPBIL consume more running time in most of the instances, compared with the other four EAs. The following explains why. nEDA and IEPBIL both belong to EDA that combines machine learning and evolutionary search. By evolving one or more probability vectors, an EDA can guide the search by exploring a number of promising areas in the search space in parallel and track global optima. For ordinary EAs, fitness evaluation is regarded as the most time-consuming procedure during evolution. However, for EDAs, apart from fitness evaluation, updating probability vector(s) is also non-trivial in terms of the computational cost. This is the reason why nEDA and IEPBIL usually lead to a heavier computational burden than RA-GA, AP-ACO, SSA, and CGA. Fortunately, nEDA has an acceptable ACT performance, given its excellent performance on all the test instances regarding the solution quality.

5.3. Performance evaluation of the two-stage approach

To evaluate the performance of the proposed two-stage approach for the MVNFP problem (TS-M), we compare it against five state-of-the-art approximation and heuristic algorithms specially devised for addressing the MVNFP problem, as listed in Table 9. Note that we use Eq. (10) as the cost function of each algorithm to make a fair comparison. All results are collected by running each algorithm 20 times on a machine with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz and 16 GB RAM.

Fig. 14 shows the average cost values obtained by the six algorithms in 18 test instances. It is clearly seen that TS-M achieves the best performance as it obtains the least average cost in all test instances. TSA and ANMP are the second and third best algorithms. Between TSA and ANMP, the former is a winner in 11 instances while it is beaten by the latter in 7 instances. Their underlying principles are similar. TSA uses a multilevel overlay-directed network to generate solutions while ANMP adopts an auxiliary graph to construct forwarding paths with VNFs hosted. In TSA, a multipath service function tree is built after all optimal unicast paths are obtained. This, to a certain extent, deteriorates the performance of TSA since global exploration is not fully considered. In ANMP, the compute resource overloading problem usually incurs, which is likely to result in infeasible solutions and poor optimization performance.

SFMP is the fourth-best algorithm since it outperforms ACMP and MMTC in most instances. However, the performance of SFMP is not stable. For example, in some cases, such as I-1, I-4, and I-13, SFMP does not achieve decent performance. This is because SFMP adopts Prim's algorithm to construct a minimum spanning tree. However, bandwidth restriction is not considered, which results in deteriorated performance.

ACMP and MMTC are the two worst algorithms considering all test instances. In ACMP, a conservative request strategy is adopted, where candidate nodes for VNFP have to meet all stringent resource requirements. In MMTC, a spanning tree originates from a node hosting the last VNF in a given SFC as the source. This is not beneficial to finding global optima since such a spanning tree is constructed based on a partial network topology only. Without the whole network considered, the resultant solutions are often local optima.

The average cost values obtained by the six algorithms are collected in Table 10. Obviously, TS-M outperforms all algorithms in each test instance. In addition, the corresponding box-plots on 18 test instances are illustrated in Fig. 15, which also indicates that TS-M can generate better solutions than SFMP, TSA, ANMP, ACMP, and MMTC.

Table 11 shows the Student's t-test results of the six algorithms. TS-M outperforms SFMP, TSA, ANMP, ACMP, and MMTC in 15 instances except I-3, I-5, and I-13. TS-M is statistically

Table 8
Results of ACT (Sec.).

Algorithm	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9
nEDA	45.53	15.59	175.03	22.31	50.75	8.16	15.98	21.67	34.81
IEPBIL	44.31	14.45	111.15	18.46	37.35	15.04	13.98	10.82	27.40
RA-GA	47.71	12.43	91.42	14.60	23.45	8.25	12.22	9.57	16.39
AP-ACO	15.53	16.93	60.47	7.04	18.66	7.70	10.02	9.91	14.49
SSA	10.03	12.38	38.35	6.28	12.43	8.37	10.41	7.50	13.14
CGA	29.28	8.32	105.74	15.95	25.21	5.70	7.83	7.04	13.15
Algorithm	I-10	I-11	I-12	I-13	I-14	I-15	I-16	I-17	I-18
nEDA	49.81	13.84	27.63	34.63	24.74	37.12	36.88	32.71	39.16
IEPBIL	45.53	12.91	21.45	24.94	21.39	29.39	25.80	25.24	21.46
RA-GA	53.31	11.26	12.71	16.04	18.03	20.12	16.47	16.02	16.27
AP-ACO	15.01	11.91	10.24	12.56	15.04	17.77	13.03	13.37	14.73
SSA	11.89	11.06	10.63	13.26	14.49	16.83	13.21	13.12	13.17
CGA	41.60	7.60	8.80	11.58	11.77	13.93	12.54	12.62	13.31

Table 9
Description of the five state-of-the-art approximation and heuristic algorithms.

Algorithm	Author	Description
SFMP	Yi et al. [39]	The heuristic algorithm for the static service function multicast problem. The propagation delay is regarded as the link cost in our experiment. If the shortest path between an access point and its associated node hosting VNF(s) does not satisfy Inequality (6), we replace it with a randomly selected link with sufficient bandwidth.
TSA	Ren et al. [12]	The two-stage heuristic algorithm for the service function tree embedding problem. Since the original TSA does not specify what the link cost is, we use propagation delay as the link cost. Besides, bandwidth was not considered before. So, we randomly choose feasible links to replace illegal ones. The original TSA does not consider the overloading problem. In our experiment, we randomly select feasible nodes when overloading occurs.
ANMP	Xu et al. [37]	The approximation algorithm for the NFV-enabled multicast problem in the mobile edge cloud environment. The propagation delay is used as the link cost. We randomly select feasible links to replace those who do not meet Inequalities (3) and (6).
ACMP	Ma et al. [8]	The approximation algorithm for the least-cost single NFV-enabled multicast problem. The original ACMP does not take Inequality (6) into account when finding an auxiliary directed acyclic graph. In our experiment, any link without sufficient bandwidth is replaced with a randomly selected one with enough bandwidth.
MMTC	Xie et al. [14]	The heuristic algorithm for multi-source multicast routing problem in NFV. The original MMTC regards bandwidth consumption as link cost, where Inequalities (3) and (6) were not considered. In our experiment, links that cannot meet the above bandwidth constraints are replaced with those with enough bandwidth.

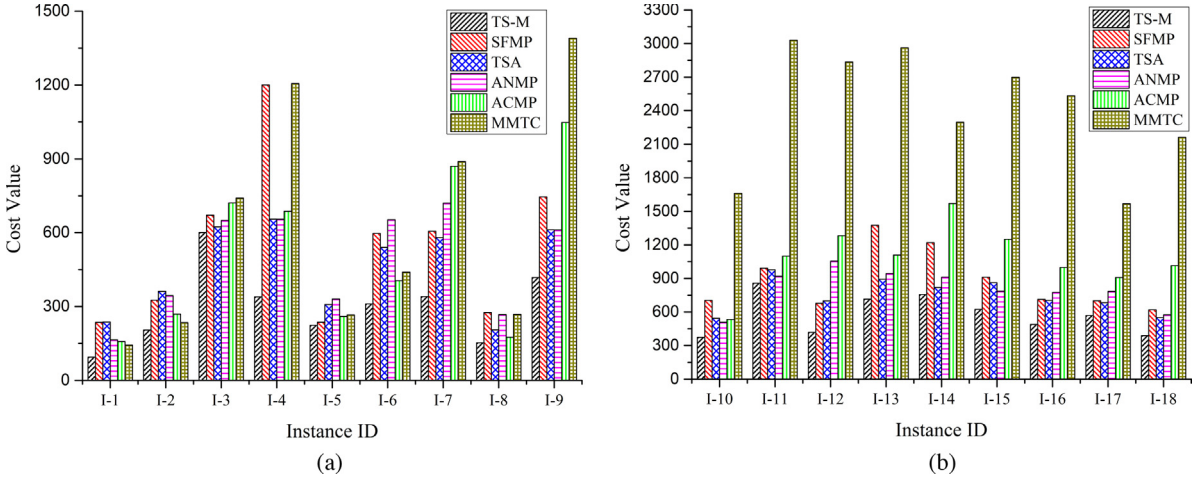

Fig. 14. Average cost values obtained in 18 test instances.

Table 10
Results of average cost values (Best results are in bold).

Algorithm	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9
TS-M	**93.69**	**204.17**	**600.41**	**339.19**	**223.23**	**310.76**	**340.17**	**152.06**	**417.46**
SFMP	235.22	325.75	671.34	1200.23	235.87	597.21	605.75	275.28	745.88
TSA	237.11	361.64	623.62	655.10	307.83	540.23	580.10	204.80	611.48
ANMP	163.86	343.97	649.62	654.51	330.27	652.67	720.03	266.91	611.25
ACMP	156.89	269.01	720.92	687.11	259.40	404.90	870.00	175.18	1047.52
MMTC	143.12	234.12	740.77	1205.49	265.55	439.60	888.99	267.68	1390.17
Algorithm	I-10	I-11	I-12	I-13	I-14	I-15	I-16	I-17	I-18
TS-M	**373.93**	**856.71**	**417.50**	**715.13**	**755.41**	**623.64**	**488.40**	**568.52**	**388.45**
SFMP	703.81	992.36	677.18	1375.06	1218.75	911.54	712.57	700.75	619.91
TSA	544.00	977.91	699.29	893.22	820.41	863.09	703.91	683.00	550.37
ANMP	508.09	919.24	1055.09	944.15	912.09	786.66	775.94	784.77	574.00
ACMP	532.82	1099.58	1281.87	1108.80	1570.82	1249.91	998.00	907.99	1014.51
MMTC	1661.19	3031.15	2835.71	2963.33	2296.19	2698.51	2533.44	1568.67	2163.16

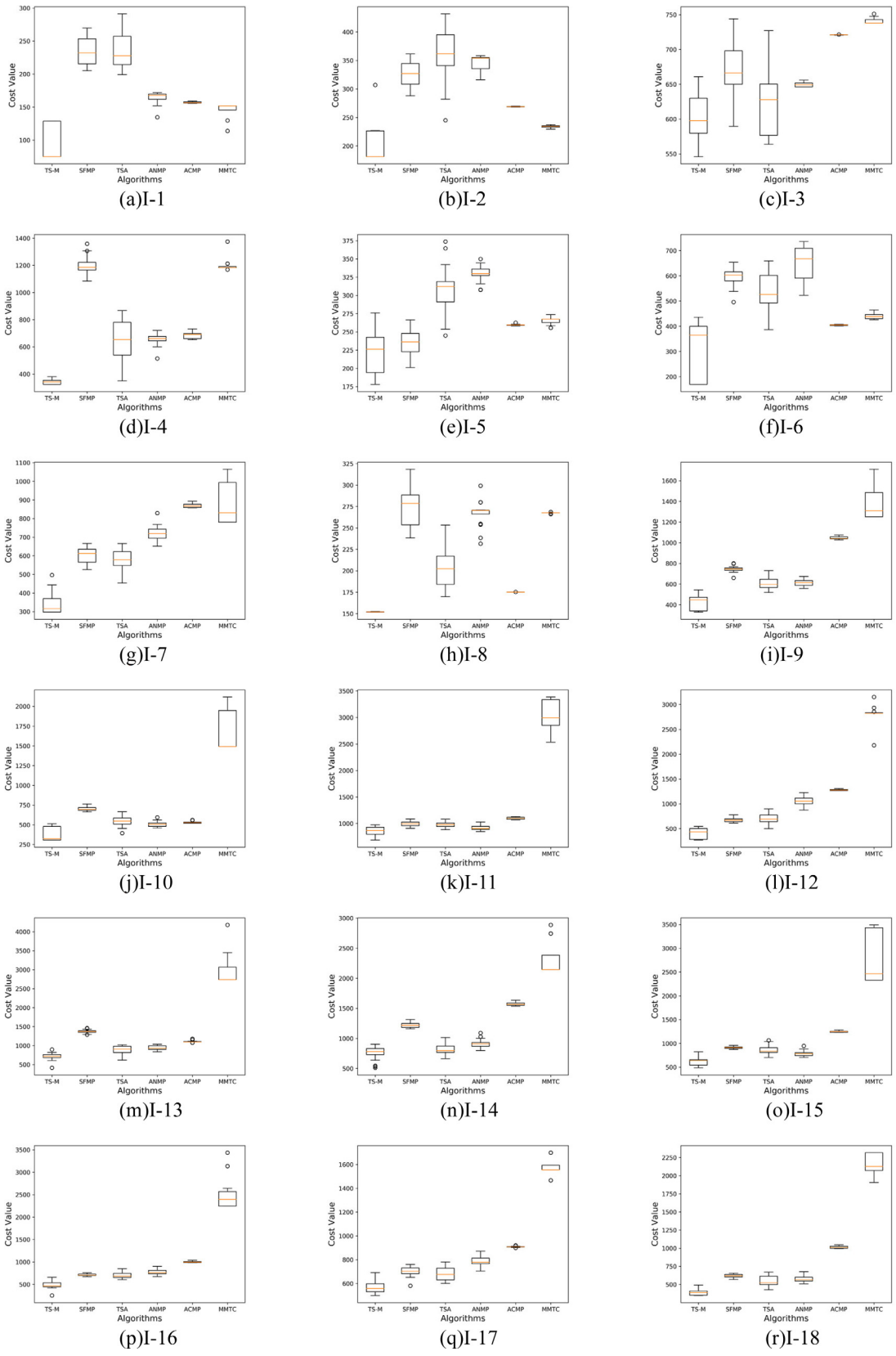

Fig. 15. Box-plots of the average cost values on 18 test instances.

equivalent to SFMP in I-5, to TSA in I-3 and I-13, respectively. Besides, the Friedman test is also used for algorithm performance comparison. With all 18 instances considered, the average rankings of the six algorithms are shown in Table 12. With the average cost, box-plot, t-test, and Friedman test results considered, one can easily conclude that TS-M is one of the most appropriate algorithms for solving the MVNFP problem.

Table 11
Results of t-test.

Alg.1 ↔ Alg.2	I-1	I-2	I-3	I-4	I-5	I-6	I-7	I-8	I-9
TS-M ↔ SFMP	+	+	+	+	∼	+	+	+	+
TS-M ↔ TSA	+	+	∼	+	+	+	+	+	+
TS-M ↔ ANMP	+	+	+	+	+	+	+	+	+
TS-M ↔ ACMP	+	+	+	+	+	+	+	+	+
TS-M ↔ MMTC	+	+	+	+	+	+	+	+	+
Alg.1 ↔ Alg.2	I-10	I-11	I-12	I-13	I-14	I-15	I-16	I-17	I-18
TS-M ↔ SFMP	+	+	+	+	+	+	+	+	+
TS-M ↔ TSA	+	+	+	∼	+	+	+	+	+
TS-M ↔ ANMP	+	+	+	+	+	+	+	+	+
TS-M ↔ ACMP	+	+	+	+	+	+	+	+	+
TS-M ↔ MMTC	+	+	+	+	+	+	+	+	+

Table 12
Rankings of six approximation and heuristic algorithms.

Algorithm	TS-M	SFMP	TSA	ANMP	ACMP	MMTC
Average rank	1	3.94	3.17	3.61	4.11	5.17
Position	1	4	2	3	5	6

6. Conclusion and future work

This paper formulates a variant of the multicast-oriented virtual network function placement (MVNFP) problem. The compute resource consumption and end-to-end delay are aggregated into a single objective function, with the bandwidth consumption constrained. This problem is divided into the multicast construction sub-problem and the VNFP sub-problem. A two-stage approach is devised to address it. Dijkstra's algorithm is applied to solve the first sub-problem, while nEDA is designed to tackle the second one. nEDA is featured with a 2-dimensional problem-specific solution encoding scheme suitable for large-scale networks, three probability vectors for solution generation and a flexible probability vectors update scheme in favor of population diversification. Simulation results show that nEDA performs better than a number of evolutionary algorithms, including population-based incremental learning, traditional and cultural genetic algorithms, ant colony optimization, and salp swarm algorithm, with respect to the convergence curve, average best fitness value, as well as results of box-plot, t-test, Friedman test, and average computational time. In addition, the two-stage approach overweighs a number of state-of-the-art approximation and heuristic algorithms in terms of the average cost value, and results of box-plot, t-test, and Friedman test indicating the effectiveness of our algorithm.

This paper considers the end-to-end delay, compute resource, and bandwidth consumption in the MVNFP problem. However, reliability, load balancing, and packet loss rate are also important QoS parameters. Considering more QoS parameters helps to improve the user experience [72]. This is included in our short-term research plan. On the other hand, the proposed two-stage approach is devised to work in a static environment. However, real-world communications networks are dynamic and full of uncertainty, e.g., multicast service requests may change over time as user demands vary. A static approach cannot meet an ever-changing demand for network service. Thus, we plan to design efficient online algorithms for the dynamic MVNFP problem. One possible solution is to adopt deep learning technology to predict the upcoming multicast service requests. Based on the prediction results, we generate the MVNFP solutions in advance.

CRediT authorship contribution statement

Xinhan Wang: Conceptualization, Methodology, Writing – original draft. **Huanlai Xing:** Conceptualization, Methodology, Writing – review & editing. **Dawei Zhan:** Methodology, Software. **Shouxi Luo:** Investigation, Visualization. **Penglin Dai:** Software, Validation. **Muhammad Azhar Iqbal:** Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (No. 61802319, No. 62002300), China Postdoctoral Science Foundation (No. 2019M660245, No. 2019M663552, No. 2020T130547), the Fundamental Research Funds for the Central Universities, and China Scholarship Council, P.R. China.

References

[1] R. Malli, X. Zhang, C. Qiao, Benefit of multicasting in all-optical networks, in: 1998 SPIE All-Optical Networking: Architecture, Control and Management Issues, 1998, pp. 209–220.
[2] A. Muhammad, L. Qu, C. Assi, Delay-aware multi-source multicast resource optimization in NFV-enabled network, in: 2020 IEEE International Conference on Communications, ICC, Dublin, Ireland, 2020, pp. 1–7.
[3] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: challenges and opportunities for innovations, IEEE Commun. Mag. 53 (2) (2015) 90–97.
[4] F. Schardong, I. Nunes, A. Schaeffer-Filho, NFV Resource allocation: a systematic review and taxonomy of VNF forwarding graph embedding, Comput. Netw. 185 (2021) 107726.
[5] J. Matias, J. Garay, N. Toledo, J. Unzilla, E. Jacob, Toward an SDN-enabled NFV architecture, IEEE Commun. Mag. 53 (4) (2015) 187–193.
[6] A.-S. Charismiadis, D. Tsolkas, N. Passas, L. Merakos, A metaheuristic approach for minimizing service creation time in slice-enabled networks, in: 2020 IEEE International Conference on Communications, ICC, Dublin, Ireland, 2020, pp. 1–6.
[7] J.S. Pujol Roig, D.M. Gutierrez-Estevez, D. Gündüz, Management and orchestration of virtual network functions via deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2020) 304–317.
[8] Y. Ma, W. Liang, J. Wu, Z. Xu, Throughput maximization of NFV-enabled multicasting in mobile edge cloud networks, IEEE Trans. Parallel Distrib. Syst. 31 (2) (2020) 393–407.
[9] R. Cohen, L. Lewin-Eytan, J.S. Naor, D. Raz, Near optimal placement of virtual network functions, in: 2015 IEEE Conference on Computer Communications, INFOCOM, Kowloon, Hong Kong, 2015, pp. 1346–1354.
[10] S. Demirci, S. Sagiroglu, Optimal placement of virtual network functions in software defined networks: a survey, J. Netw. Comput. Appl. 147 (2019) 102424.
[11] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: state-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262.
[12] B. Ren, D. Guo, Y. Shen, G. Tang, X. Lin, Embedding service function tree with minimum cost for NFV-enabled multicast, IEEE J. Sel. Areas Commun. 37 (5) (2019) 1085–1097.
[13] J.-J. Kuo, S.-H. Shen, M.-H. Yang, D.-N. Yang, M.-J. Tsai, W.-T. Chen, Service overlay forest embedding for software-defined cloud networks, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, Atlanta, GA, USA, 2017, pp. 720–730.
[14] K. Xie, X. Zhou, T. Semong, S. He, Multi-source multicast routing with QoS constraints in network function virtualization, in: 2019 IEEE International Conference on Communications, ICC, Shanghai, China, 2019, pp. 1–6.
[15] P. Veitch, M.J. McGrath, V. Bayon, An instrumentation and analytics framework for optimal and robust NFV deployment, IEEE Commun. Mag. 53 (2) (2015) 126–133.
[16] K.-L. Du, M.N.S. Swamy, Search and Optimization by Metaheuristics, Springer International Publishing, Switzerland, 2016.
[17] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm Evol. Comput. 1 (3) (2011) 111–128.
[18] W. Shao, D. Pi, Z. Shao, A Pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow-shop scheduling problem with sequence-dependent setup time, IEEE Trans. Autom. Sci. Eng. 16 (3) (2019) 1344–1360.

[19] S.N. Makhadmeh, A.T. Khader, M.A. Al-Betar, S. Naim, A.K. Abasi, Z.A.A. Alyasseri, Optimization methods for power scheduling problems in smart home: survey, Renew. Sust. Energ. Rev. 115 (2019) 109362.

[20] R. Pérez-Rodríguez, A. Hernández-Aguirre, A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows, Comput. Ind. Eng. 130 (2019) 75–96.

[21] Y. Xiang, X. Yang, Y. Zhou, H. Huang, Enhancing decomposition-based algorithms by estimation of distribution for constrained optimal software product selection, IEEE Trans. Evol. Comput. 24 (2) (2020) 245–259.

[22] Z. Wang, M. Gong, Dynamic deployment optimization of near space communication system using a novel estimation of distribution algorithm, Appl. Soft Comput. 78 (2019) 569–582.

[23] X. Wang, H. Zhao, T. Han, Z. Wei, Y. Liang, Y. Li, A Gaussian estimation of distribution algorithm with random walk strategies and its application in optimal missile guidance handover for multi-UCAV in over-the-horizon air combat, IEEE Access. 7 (2019) 43298–43317.

[24] M. Faraji Amiri, J. Behnamian, Multi-objective green flowshop scheduling problem under uncertainty: estimation of distribution algorithm, J. Clean. Prod. 251 (2020) 119734.

[25] Y. Chen, Y. Jin, X. Sun, Language model based interactive estimation of distribution algorithm, Knowl.-Based Syst. 200 (2020) 105980.

[26] Z.-G. Chen, Y. Lin, Y.-J. Gong, Z.-H. Zhan, J. Zhang, Maximizing lifetime of range-adjustable wireless sensor networks: a neighborhood-based estimation of distribution algorithm, IEEE Trans. Cybern. (2020) 1–12.

[27] N. Kiji, T. Sato, R. Shinkuma, E. Oki, Virtual network function placement and routing model for multicast service chaining based on merging multiple service paths, in: 2019 IEEE 20th International Conference on High Performance Switching and Routing, HPSR, Xi'An, China, 2019, pp. 1–6.

[28] Y. Qin, Q. Xia, Z. Xu, P. Zhou, A. Galis, O.F. Rana, J. Ren, G. Wu, Enabling multicast slices in edge networks, IEEE Internet Things J. 7 (2020) 8485–8501.

[29] S.Q. Zhang, Q. Zhang, H. Bannazadeh, A. Leon-Garcia, Routing algorithms for network function virtualization enabled multicast topology on SDN, IEEE Trans. Netw. Serv. Manag. 12 (4) (2015) 580–594.

[30] S.Q. Zhang, Q. Zhang, H. Bannazadeh, A. Leon-Garcia, Network function virtualization enabled multicast routing on SDN, in: 2015 IEEE International Conference on Communications, ICC, London, UK, 2015, pp. 5595–5601.

[31] S.Q. Zhang, A. Tizghadam, B. Park, H. Bannazadeh, A. Leon-Garcia, Joint NFV placement and routing for multicast service on SDN, in: 2016 IEEE/IFIP Network Operations and Management Symposium, NOMS, Istanbul, Turkey, 2016, pp. 333–341.

[32] X. Gao, W. Zhong, Z. Ye, Y. Zhao, J. Fan, X. Cao, H. Yu, C. Qiao, Virtual network mapping for reliable multicast services with max-min fairness, in: 2015 IEEE Global Communications Conference, GLOBECOM, San Diego, CA, USA, 2015, pp. 1–6.

[33] M. Zeng, W. Fang, J.J.P.C. Rodrigues, Z. Zhu, Orchestrating multicast-oriented NFV trees in inter-DC elastic optical networks, in: 2016 IEEE International Conference on Communications, ICC, Kuala Lumpur, Malaysia, 2016, pp. 1–6.

[34] O. Alhussein, P.T. Do, J. Li, Q. Ye, W. Shi, W. Zhuang, X. Shen, X. Li, J. Rao, Joint VNF placement and multicast traffic routing in 5G core networks, in: 2018 IEEE Global Communications Conference, GLOBECOM, Abu Dhabi, United Arab Emirates, 2018, pp. 1–6.

[35] L. Qu, C. Assi, Reliability-aware multi-source multicast hybrid routing in softwarized networks, IEEE Access 8 (2020) 113331–113341.

[36] Z. Xu, W. Liang, M. Huang, M. Jia, S. Guo, A. Galis, Efficient NFV-enabled multicasting in SDNs, IEEE Trans. Commun. 67 (3) (2019) 2052–2070.

[37] Z. Xu, Y. Zhang, W. Liang, Q. Xia, O. Rana, A. Galis, G. Wu, P. Zhou, NFV-Enabled multicasting in mobile edge clouds with resource sharing, in: Proceedings of the 48th International Conference on Parallel Processing, ACM, Kyoto, Japan, 2019, pp. 1–10.

[38] B. Yi, X. Wang, M. Huang, A. Dong, A multi-stage solution for NFV-enabled multicast over the hybrid infrastructure, IEEE Commun. Lett. 21 (9) (2017) 2061–2064.

[39] B. Yi, X. Wang, M. Huang, L. Ma, SDN and NFV enabled service function multicast mechanisms over hybrid infrastructure, Peer–Peer Netw. Appl. 12 (2019) 405–421.

[40] H. Soni, W. Dabbous, T. Turletti, H. Asaeda, NFV-based scalable guaranteed-bandwidth multicast service for software defined ISP networks, IEEE Trans. Netw. Serv. Manag. 14 (4) (2017) 1157–1170.

[41] Y. Cheng, L. Yang, VNF deployment and routing for NFV-enabled multicast: a Steiner tree-based approach, in: 2017 9th International Conference on Wireless Communications and Signal Processing, WCSP, IEEE, Nanjing, China, 2017, pp. 1–4.

[42] K. Zahoor, K. Bilal, A. Erbad, A. Mohamed, M. Guizani, Multicast at edge: an edge network architecture for service-less crowdsourced live video multicast, IEEE Access 9 (2021) 59508–59526.

[43] X. Wang, H. Xing, H. Yang, On multicast-oriented virtual network function placement: A modified genetic algorithm, in: International Conference on Signal and Information Processing, Networking and Computers, Springer, Yuzhou, China, 2019, pp. 420–428.

[44] H. Xing, X. Zhou, X. Wang, S. Luo, P. Dai, K. Li, H. Yang, An integer encoding grey wolf optimizer for virtual network function placement, Appl. Soft Comput. 76 (2019) 575–594.

[45] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.

[46] B. Zhang, H.T. Mouftah, A destination-driven shortest path tree algorithm, in: 2002 IEEE International Conference on Communications, ICC, New York, NY, USA, 2002, pp. 2258–2262.

[47] H. Xing, S. Li, Y. Cui, L. Yan, W. Pan, R. Qu, A hybrid EDA for load balancing in multicast with network coding, Appl. Soft Comput. 59 (2017) 363–377.

[48] J. Lee, I. Yu, J. Park, D.-W. Kim, Memetic feature selection for multilabel text categorization using label frequency difference, Inform. Sci. 485 (2019) 263–280.

[49] G. Povéda, O. Regnier-Coudert, F. Teichteil-Königsbuch, G. Dupont, A. Arnold, J. Guerra, M. Picard, Evolutionary approaches to dynamic earth observation satellites mission planning under uncertainty, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, ACM, Prague, Czech Republic, 2019, pp. 1302–1310.

[50] J. Li, X. Meng, M. Zhou, X. Dai, A two-stage approach to path planning and collision avoidance of multibridge machining systems, IEEE Trans. Syst. Man Cybern. Syst. 47 (7) (2017) 1039–1049.

[51] H. Xing, S. Li, P. Dai, S. Luo, K. Li, H. Yang, PBIL for delay constrained virtual network function placement with load balancing, in: Proceedings of the ACM Turing Celebration Conference-China, ACM, Chengdu, China, 2019, pp. 1–6.

[52] S.-H. Chen, M.-C. Chen, P.-C. Chang, Q. Zhang, Y.-M. Chen, Guidelines for developing effective estimation of distribution algorithms in solving single machine scheduling problems, Expert Syst. Appl. 37 (9) (2010) 6441–6451.

[53] H. Handa, The effectiveness of mutation operation in the case of estimation of distribution algorithms, Biosystems 87 (2–3) (2007) 243–251.

[54] SNDlib, 2020, 30 July, URL: http://sndlib.zib.de.

[55] The Internet Topology Zoo, 2020, 30 July, URL: http://www.topology-zoo.org.

[56] V. Batagelj, U. Brandes, Efficient generation of large random networks, Phys. Rev. E 71 (3) (2005) 036113.

[57] N. Singh, L.H. Son, F. Chiclana, J.-P. Magnot, A new fusion of salp swarm with sine cosine for optimization of non-linear functions, Eng. Comput. 36 (2020) 185–212.

[58] D. Jiao, L. Ke, W. Yang, J. Li, An estimation of distribution algorithm based load-balanced clustering of wireless sensor networks, in: 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 2017, pp. 151–158.

[59] F. Carpio, S. Dhahri, A. Jukan, VNF placement with replication for load balancing in NFV networks, in: 2017 IEEE International Conference on Communications, ICC, Paris, France, 2017, pp. 1–6.

[60] W. Wei, H. Gu, W. Lu, T. Zhou, X. Liu, Energy efficient virtual machine placement with an improved ant colony optimization over data center networks, IEEE Access 7 (2019) 60617–60625.

[61] L. Abualigah, M. Shehab, M. Alshinwan, H. Alabool, Salp swarm algorithm: a comprehensive survey, Neural Comput. Appl. 32 (2020) 11195–11215.

[62] L. Laaziz, N. Kara, R. Rabipour, C. Edstrom, Y. Lemieux, FASTSCALE: A fast and scalable evolutionary algorithm for the joint placement and chaining of virtualized services, J. Netw. Comput. Appl. 148 (2019) 102429.

[63] R.E. Walpole, R.H. Myers, S.L. Myers, K. Ye, Probability and Statistics for Engineers and Scientists, 8th ed., Pearson Education, Boston, US, 2007.

[64] S.M. Ross, Introduction To Probability and Statistics for Engineers and Scientists, 5th ed., Academic Press, London, UK, 2014.

[65] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai, R. Qu, A multiobjective computation offloading algorithm for mobile edge computing, IEEE Internet Things J. 7 (2020) 8780–8799.

[66] Y. Chen, Y. Jin, X. Sun, Language model based interactive estimation of distribution algorithm, Knowl.-Based Syst. 200 (2020) 105980.

[67] H. Faris, M.M. Mafarja, A.A. Heidari, I. Aljarah, A.M. Al-Zoubi, S. Mirjalili, H. Fujita, An efficient binary salp swarm algorithm with crossover scheme for feature selection problems, Knowl.-Based Syst. 154 (2018) 43–67.

[68] N. Singh, L.H. Son, F. Chiclana, J.-P. Magnot, A new fusion of salp swarm with sine cosine for optimization of non-linear functions, Eng. Comput. 36 (2020) 185–212.

[69] S.S. Alresheedi, S. Lu, M. Abd Elaziz, A.A. Ewees, Improved multiobjective salp swarm optimization for virtual machine placement in cloud computing, Hum.-Centric Comput. Inf. Sci. 9 (1) (2019) 15.

[70] A. Farshin, S. Sharifian, A modified knowledge-based ant colony algorithm for virtual machine placement and simultaneous routing of NFV in distributed cloud architecture, J. Supercomput. 75 (2019) 5520–5550.

[71] F. Alharbi, Y.-C. Tian, M. Tang, W.-Z. Zhang, C. Peng, M. Fei, An ant colony system for energy-efficient dynamic virtual machine placement in data centers, Expert Syst. Appl. 120 (2019) 228–238.

[72] P.P. Ray, N. Kumar, SDN/NFV architectures for edge-cloud oriented IoT: a systematic review, Comput. Commu. 169 (2021) 129–153.