Mobile Networks and Applications (2020) 25:2191–2205
https://doi.org/10.1007/s11036-020-01661-w

Service Function Chain Placement for Joint Cost and Latency Optimization

Mohammad Ali Khoshkholghi[1] · Michel Gokan Khan[1] · Kyoomars Alizadeh Noghani[1] · Javid Taheri[1] · Deval Bhamare[1] · Andreas Kassler[1] · Zhengzhe Xiang[2] · Shuiguang Deng[2] · Xiaoxian Yang[3]

Accepted: 23 September 2020 / Published online: 21 November 2020

Abstract
Network Function Virtualization (NFV) is an emerging technology to consolidate network functions onto high volume storages, servers and switches located anywhere in the network. Virtual Network Functions (VNFs) are chained together to provide a specific network service, called Service Function Chains (SFCs). Regarding to Quality of Service (QoS) requirements and network features and states, SFCs are served through performing two tasks: VNF placement and link embedding on the substrate networks. Reducing deployment cost is a desired objective for all service providers in cloud/edge environments to increase their profit form demanded services. However, increasing resource utilization in order to decrease deployment cost may lead to increase the service latency and consequently increase SLA violation and decrease user satisfaction. To this end, we formulate a multi-objective optimization model to joint VNF placement and link embedding in order to reduce deployment cost and service latency with respect to a variety of constraints. We, then solve the optimization problem using two heuristic-based algorithms that perform close to optimum for large scale cloud/edge environments. Since the optimization model involves conflicting objectives, we also investigate pareto optimal solution so that it optimizes multiple objectives as much as possible. The efficiency of proposed algorithms is evaluated using both simulation and emulation. The evaluation results show that the proposed optimization approach succeed in minimizing both cost and latency while the results are as accurate as optimal solution obtained by Gurobi (5%).

Keywords Cloud/edge computing · Network function virtualization · Optimization · Service chain placement

1 Introduction

NFV is an innovational network architecture to provide network services by decoupling network functions such as firewalls, intrusion detection, load balancing and routing from physical boxes so that they can run as software-based applications. Therefore, it can improve flexibility and agility of the network since it is easier to dynamically scale the VNF instances, send the functions across a distributed infrastructure and upgrade the software without interrupting the service. In addition, NFV enables VNFs to be placed on cloud/edge physical machines in the form of virtual machine (VM) or other containers such as Linux container and Docker [1] in order for parallel processing of multiple operations, load balancing between servers, decreasing the traffic congestion and locating closer to the end users [2]. VNFs are chained together in a predefined order to make particular network services. Service flow traffics should be traversed through the relevant SFCs while satisfying business requirements and system constraints. SFCs are considered as virtual networks including VNFs and virtual links between them which can be mapped on the substrate network. The link mapping and VNF placement should be performed regarding the demanded resources and can affect cost of services requested by users.

Over the past few years, user traffic and the use of virtualization technologies have been growing very fast in communication networks. The excessive needs for developing new services and deployment of the required network resources as well as maintaining, upgrading and expanding physical infrastructures emerge a remarkable operational expenditure (OPEX) and capital expenditure (CAPEX) for the

✉ Mohammad Ali Khoshkholghi
ali.khosh-kholghi@kau.se

[1] Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden

[2] College of Computer Science and Technology, Zhejiang University, Hangzhou, China

[3] School of Computer and Information Engineering, Shanghai Ploytechnic University, Shanghai, China

network service providers. NFV is able to significantly decrease the capital and operational cost and outperform resource allocation more efficient and flexible. An efficient deployment of SFCs plays a major rule in decreasing monetary costs of the service providers. However, sometimes, there would be a tradeoff between different objectives as they may be contradictory. For example, although increasing the resource utilization in order to minimizing the number of active physical nodes can reduce deployment cost, but at the same time it may lead to an increase in latency of demanded services. It can happen since the aggregation traffic on the physical nodes and links increases and it degrades the efficiency of latency objective. On the other side, minimizing network latency can be resulted in an increase in deployment and network cost because of more resources needed for providing services [3].

Figure 1 illustrates an example of VNF placement and link embedding on a small substrate network including 10 cloud/edge physical nodes. As we can see in Table 1, there are 3 SFCs to be placed on the substrate network. The first service chain includes three VNFs which are placed on nodes 3, 4 and 9, and the virtual links connecting the VNFs are mapped on the given physical path (3–4–7-9). Similarly, the second service chain including f_4 and f_5 is placed on nodes 6 and 8 through intermediate node 10. For the last service chain, physical nodes 9, 8, 5 and 2 are selected to host the VNFs with the selected physical path in the sequence of nodes 9,10,8,5,2.

In this paper, we propose a multi-objective optimization model to minimize joint deployment cost of service providers and end-to-end latency of SFCs. We formulate the problem of SFC placement using Mixed Integer Programming (MIP) model which takes into account a range of constraints such as resource capacity, acceptable latency requirements, affinity and anti-affinity. The proposed model enables service providers to accept and serve more user requests with strict latency needs while keeping overall costs low.

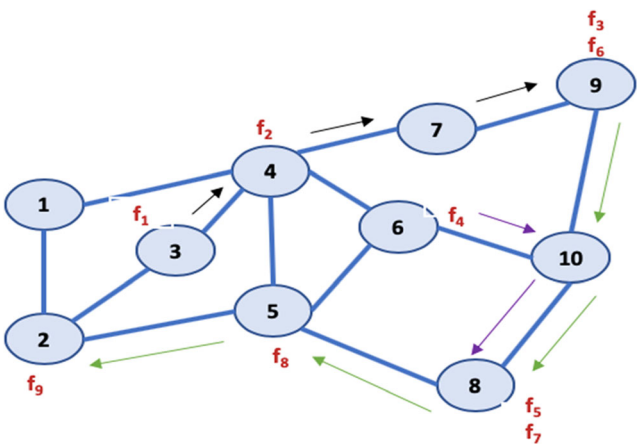

Fig. 1 An example to illustrate VNF placement and link embedding in cloud/edge networks

Table 1 SFC mapping on the substrate network

SFC index	Virtual path	Physical path
1	f_1 - f_2 - f_3	v_3–v_4– v_7–v_9
2	f_4 - f_5	v_6–v_{10}– v_8
3	f_6 - f_7 - f_8 - f_9	v_9–v_{10}– v_8–v_5–v_2

As finding an optimum solution for service chain placement is a NP-hard problem, exact solutions needs huge amount of time and computational resources, so they are impractical for the real-world networks. Heuristic algorithms have been proposed to improve the scalability of solution and to handle the large infrastructures. These approaches have to make a trade-off between optimality of the solution, complexity and execution time. We propose two genetic-based and bee colony-based algorithms which can be used for large networks to place service functions and routing simultaneously, as it increases the performance of virtual network embedding mechanism.

Although much research attention is given to VNF placement, as far as we know, none of the previous works investigated the optimization problem of joint deployment cost and end-to-end latency of SFCs to find a pareto optimal solution which can be used to make a tradeoff between monetary cost and network delay cost in cloud/edge service providers. In addition, for the sake of achieving an efficient and accurate solution needed for real-world networks, we propose comprehensive cost and latency models taking into consideration different effective factors as energy consumption, software license, computation resources and network usage (to define cost model), as well as queuing delay, processing and virtualization delay, and propagation delay (to define latency model).

The main contributions of this paper are summarized as follows:

1. We propose a multi-objective optimization model formulated as MIP, to jointly minimize the deployment cost and the end-to-end latency regarding a range of constraints such as routing, capacity, delay, location constraints.
2. We propose a genetic-based algorithm as well as a bee colony-based algorithm to solve the placement and routing problem to find a pareto optimal solution considering heterogenous physical nodes and various SFCs.
3. We evaluate our approach using both simulation and emulation (Mininet), and compare the results with a greedy algorithm as well as with the optimal solution.

The rest of the paper is organized as follows. The related works are discussed in Section 2. In Section 3, we formulate the problem of SFC placement in order to jointly minimize the deployment cost and network latency. In Section 4, we illustrate our proposed algorithms to solve the optimization problem in details. Section 5 provides the performance evaluation of the proposed algorithms compared with the benchmark algorithms. Finally, we conclude the paper in section 6.

2 Related works

VNF placement has recently obtained much attention in the literature. The related works can be categorized regarding various aspects such as system models, objectives and proposed solutions. A number of studies have formulated the placement problem as integer programming problems, and solved them using optimization solvers; or applied heuristics or greedy approaches to place VNFs. Bari et al. [4] proposed a Mixed Integer Linear Programming (MILP) model to optimize utilization and to reduce network operational costs, and solved it using CPLEX. Authors in [5] proposed an ILP formulation to decrease the number of deployed instances. In [6, 7], the authors proposed a MILP and Mixed Integer Quadratically Constrained Program (MIQCP) for VNF placement in data centers, respectively. These approaches are effective to find the exact or near-optimal solutions, but only for small size infrastructures and are not suitable for the networks including large set of physical nodes. A polynomial complexity heuristic has been proposed for VNF placement in [8]. The placement problem is modeled as a Multi-Stage directed graph with associated costs and then, VNFs are placed leveraging a Viterbi algorithm [5]. Agrawal et al. [9] introduced .a .queuing-based .system model along with a heuristic to minimize the network latency, but they considered only CPU in their work and neglected other resources. The authors in [10–12] proposed QoS prediction strategies for mobile ecommerce environments, edge environments and IoT services, respectively.

In respect of objective, some other works address the VNF placement regarding load balancing, service resilience and monitoring overhead [13–16]. Authors in [17] presented an offloading decision problem as a cost-minimization problem. In [18], the authors presented an energy-aware placement for NFV environments using a game-theory approach. However, they proposed the solution for small infrastructures including only a few physical nodes. In [19], the authors proposed a data-intensive service edge deployment scheme based on Genetic Algorithm in order to minimize the response time of the services. In [20], the authors encoded the state of the service provisioning system and the resource allocation scheme, and then modeled the adjustment of allocated resources for services as Markov Decision Process (MDP) in order to improve trustworthiness in IoT environments.

One of the main objectives to solve the problem of VNF placement is reducing the deployment cost [21, 22]. Deployment cost is defined as the cost of the computing and communication resources which are needed to execute a service chain. The main issue of the problem is how to place the VNFs onto physical nodes in a way that QoS is satisfied and the cost of service provider is reduced. Xia et al. [23] study the on-demand VNF placement to minimize the cost by placing the VNFs onto fewer physical nodes. Cohen et al. [24] address the VNF placement for the purpose of minimizing the deployment cost. They showed that an optimal placement can improve network reliability, performance of the network and also operation cost. However, the precedence constraint for VNFs in the service chains is neglected in this work. The authors in [25] address the problem of VNFs mapping and scheduling. They have proposed three greedy algorithms and a Tabu search algorithm to minimize the cost and increase the revenue. As a drawback of this work, authors consider only VNFs as nodes without any link between them; they did not consider service chain placement in their work.

An online application placement for mobile edge computing (MEC) has been proposed by Wang et al. [26]. The authors modeled the placement problem using Markov Decision Processes (MDP) and tried to reduce the state space of the problem. They derived a new MDP model in which only distance between servers and users define the states and then, they proposed a greedy algorithm to place the applications. Authors in [27] introduce an edge server placement approach to minimize the energy efficiency in MEC. They have proposed a particle swarm optimization to find the optimum solution. Another research [28] proposed a service chain placement system to minimize the queuing delay on nodes and infrastructure resources. Two techniques are proposed for the placement problem: a round-robin based heuristic, and a MIP-based optimization method. The results have been compared with the general MIP. However, this work only considers physical nodes and neglect the impact of routing on service latency.

The previous works either consider cost and latency as the separate objectives or consider latency as a constraint in optimization model. However, we propose a multi-objective optimization model considering joint cost and latency for the problem of VNF placement and link embedding in cloud edge systems. The proposed model is able to provide near-optimal solution for only cost optimization, only latency optimization or a pareto optimal

solution considering joint cost and latency. Therefore, service providers are able to set up a weight factor based on their needs to make a tradeoff between cost and latency. In addition, the previous works considered some important factors to model the cost and especially latency functions and neglected others which may lead to SLA violation in real networks. However, we propose a comprehensive model regarding various effective factors which makes this model more practical for real edge cloud systems.

3 System model and problem formulation

In this section, the service function chain placement in a cloud/edge network is formulated as an optimization problem. The goal of optimization model is mapping the service chains onto physical nodes so that cost and latency are minimized satisfying placement constraints. Tables 2 and 3 show the variables used in this paper.

We consider an edge/cloud network which consists three tiers: end users, edge servers and a cloud. The substrate network is defined as an undirected physical graph, denoted by $G = (V, E)$ where V is a set of N heterogeneous edge-cloud physical nodes denoted by $V = \{v_1, v_2, \ldots, v_N\}$ and E is a set of K physical links between nodes denoted by $E = \{e_1, e_2, \ldots, e_K\}$.Each physical node $v \in V$ is characterized by three types of resources as processing, memory and disk, and can host one or several network functions.

The set of all service chains in the system is indicated as S. Let $s \in S$ denotes a service chain consists of a list of VNFs with strict precedence connection, and is modeled as a directed graph denoted by $SC = (F, L)$ where F is the set of virtual functions and L is the set of virtual links. Each virtual function $f \in F$ is described by its processing, memory and disk demand, and each virtual link $l \in L$ is characterized by its bandwidth demand. We consider different virtual topologies consist of VNFs and virtual links in response to any type of requested service chains. Each VNF f belongs to only one service chain s and can be placed on only one physical node $v \in V$ which meets the defined constraints. Similarly, each $l \in L$ can be mapped to only one physical path P that satisfies the constraints. Finally, let binary variable X^v_{sf} indicates whether VNF $f \in F$ from service chains s can be mapped on physical node $v \in V$ ($X^v_{sf} = 1$) or not ($X^v_{sf} = 0$).

Considering the above-mentioned issues, we present an optimization model for VNF placement and link embedding problem which aims to 1) minimize deployment cost of service chains, 2) minimize end-to-end service latency, 3) minimize joint cost and latency. In the latter case, we aim to obtain a pareto optimality between cost and delay in the mentioned problem.

3.1 Minimizing the overall cost

Eq. (1) denotes overall monetary cost incurred in the system for all service chains. Overall cost is formulated based on three cost components as: cost of basic resources named $Ç(ŗ)$, cost of physical nodes called $Ç(ŋ)$, and communication cost named $Ç(ŧ)$.Since we consider physical nodes are heterogeneous in this work, placement of each VNF on a particular node may incur a different computational cost. As shown in Eq. (2), the cost of basic resources for each service chains ϵ S is formulated as the processing, memory and storage costs of the physical nodes where the service chain's functions are mapped on.

$$Ç(ŗ, ŋ, ŧ) = Ç(ŗ) + Ç(ŋ) + Ç(ŧ) \quad (1)$$

Table 2 Descriptions of Binary variables

Variable	Description
X^v_{sf}	Equal to 1,if VNF f is mapped to physical node v, otherwise 0
X^v_{kf}	Equal to 1,if VNF f of type k is mapped to physical node v, otherwise 0
y^e_{sl}	Equal to 1,if virtual link l is mapped to physical link e, otherwise 0
$y^{e_{uv}}_{l_{pq}}$	Equal to 1,if virtual link l between VNF p and VNF q is mapped to physical link e between physical node u and physical node v, otherwise 0
$d^{ŗ}_s$	Equal to 1 if service chain s is mapped to path ŗ, otherwise 0
w^{uv}_{spq}	Equal to 1 if service chain s traverses from VNF p on node u to VNF q on node v, otherwise 0
Z^{pq}_s	Equal to 1 if VNF q is the successor of VNF p in service chain s
$X^{ŗv}_{sf}$	Equal to 1 if VNF f from service chain s is mapped to physical node v from path ŗ, otherwise 0
A_{pq}	Equal to 1 if there is affinity between VNF p and VNF q, otherwise 0
AA_{pq}	Equal to 1 if there is anti-affinity between VNF p and VNF q, otherwise 0
z^v	Binary variable is equal to 1 if physical node v hosts at least one VNF f.

Table 3 Descriptions of variables

Symbol	Description
V	Set of physical nodes
S	Set of service chains
F	Set of VNFs in a service chain s
L	Set of virtual links
E	Set of physical links
K	Set of VNF types
P	Set of physical paths
m	Number of VNFs required by service chain s
cap^{v}_{cpu}	CPU capacity of physical node v
cap^{v}_{mem}	Memory capacity of physical node v
cap^{v}_{st}	Storage capacity of physical node v
cap^{e}_{bw}	Bandwidth capacity of physical link e
$co^{v_{cpu}}_{sf}$	CPU cost of physical node v used by VNF f
$co^{v_{mem}}_{sf}$	Memory cost of physical node v used by VNF f
$co^{v_{st}}_{sf}$	Storage cost of physical node v used by VNF f
$co^{e_{bw}}_{sl}$	Bandwidth cost of physical link e used by virtual link l
$\bar{l}_k$	Software license cost of a VNF instance of type k
d^{v}_{cf}	CPU demand by VNF f from physical node v
d^{v}_{mf}	Memory demand by VNF f from physical node v
d^{v}_{stf}	Storage demand by VNF f from physical node v
d^{e}_{bwl}	Bandwidth demand by virtual link l from physical link e
D^{e}_{l}	Propagation delay of physical link e
p^{min}_{v}	Minimum power consumption of physical node v
p^{max}_{v}	Maximum power consumption of physical node v
D^{v}_{vir}	Virtualization delay of physical node v
D^{s}_{end}	End-to-end latency of service chain s
D_{Max}	Latency bound for service chain s
Υ	Site license cost
φ	The price to convert the power to a monetary term
$C(r)$	Resources cost, r {CPU, memory, storage}
$C(n)$	Node cost
$C(t)$	Communication cost
$L(l)$	Propagation delay
$L(P)$	Processing and virtualization delay
$L(Q)$	Queuing delay
α	Weight factor, $\alpha \in [0, 1]$

$$C(r) = \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}\left(co^{v_{cpu}}_{sf} + co^{v_{mem}}_{sf} + co^{v_{st}}_{sf}\right).X^{v}_{sf} \quad (2)$$

Cost of physical nodes for all service chains is formulated in Eq. (3) which encompasses software license cost per site and per instance, and cost of power consumption. Software license cost per site [29] denoted by Υ ($/node) is calculated for all physical nodes hosting at least one VNF $f \in F$. z^v is a binary variable that indicates if a physical node is active. To be active ($z^v = 1$), a physical node v must host at least one VNF from any service chain, otherwise $z^v = 0$. On the other hand, X^{v}_{sf} is a binary variable that is equal to 1 if a VNF f from service chain s is allocated to the physical node v, otherwise it equals to 0. Therefore, we can summarize that if $X^{v}_{sf} = 1$, then consequently $z^v = 1$.

A major challenge in SFC placement is to control the number of active nodes in the system, since turning on all physical nodes increases the cost of electricity and has a negative effect on environment due to an increase in carbon footprint. Idle servers consume about 60% of peak power in data centers [30]. Hence, a smaller number of switched-on nodes are desired to minimize the overall cost of the system.

The cost of power consumption is calculated for all active servers in the system [31]. p^{min}_{v}, p^{max}_{v} denote minimum and maximum power consumption of server $v \in V$, respectively. Cap^{v}_{cpu} indicates the available processing capacity in server v, d^{v}_{cf} is amount of CPU demanded by a VNF $f \in F$,and φ is a constant to convert the power to a monetary term. The last but not least, software license cost per instance denoted by $\bar{l}_k$, is a cost incurred to the system due to create a new VNF instance for different types of VNF $k \in K$.

$$\begin{aligned} C(n) = {} & \Upsilon.\sum_{v\in V} z^v + \\ & \varphi.\sum_{v\in V}\left(p^{min}_{v} + (p^{max}_{v} - p^{min}_{v})\frac{1}{Cap^{v}_{cpu}}\sum_{f\in F} d^{v}_{cf}X^{v}_{f}\right).z^v \\ & + \sum_{k\in K}\sum_{v\in V}\sum_{f\in F}\bar{l}_k.X^{v}_{kf} \end{aligned} \quad (3)$$

The communication cost is the third component of overall cost defined in Eq. (4). It is formulated as the sum of bandwidth costs incurred by physical paths where virtual links are mapped on. The binary variable y^{e}_{sl} is equal to 1 if virtual link $l \in L$ from service chain s is mapped on physical link $e \in E$, otherwise $y^{e}_{sl} = 0$.

$$C(t) = \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\left(co^{e_{bw}}_{sl}.y^{e}_{sl}\right) \quad (4)$$

Combining Eq. (2–4), the overall cost model for all service chains is shown in Eq. (5).

$$\begin{aligned} C(r,n,t) = {} & \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}\left(co^{v_{cpu}}_{sf} + co^{v_{mem}}_{sf} + co^{v_{st}}_{sf}\right).X^{v}_{sf} \\ & + \Big(\Upsilon.\sum_{v\in V} z^v + \varphi.\sum_{v\in V}\Big(p^{min}_{v} \\ & + (p^{max}_{v} - p^{min}_{v})\frac{1}{Cap^{v}_{cpu}}\sum_{f\in F} d^{v}_{cf}X^{v}_{f}\Big).z^v \\ & + \sum_{k\in K}\sum_{v\in V}\sum_{f\in F}\bar{l}_k.X^{v}_{kf}\Big) \\ & + \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\left(co^{e_{bw}}_{sl}.y^{e}_{sl}\right) \end{aligned} \quad (5)$$

3.2 Minimizing end-to-end latency

VNFs, in each service chain, need to communicate with each other and forward packets according to the virtual links between them. These virtual links are mapped on the physical paths comprised of one or several physical links. Depending on the topology in a cloud/edge system, these physical connections may have different hops and different hop distance which can incur various amount of delay while processing a service chain. In the other hands, gathering VNFs on a limited number of nodes can increase traffic congestion in the network and increase latency of service chains. Based on the expected QoS and Service Level Agreement (SLA) between users and service providers, each service chain may have a deadline to be executed. Moreover, latency-critical services like remote surgery require stringent latency requirements in NFV infrastructures.

In this section, we propose a model which aims to minimize the end-to-end latency of service chains in the system. As we can see in Eq. (6), the latency model is formulated as propagation delay $\mathcal{L}(\text{ł})$, processing and virtualization delay $\mathcal{L}(\acute{\text{P}})$, and queuing delay $\mathcal{L}(Q)$ Neglecting each of the delay functions may result in violation of the end-to-end latency requirements.

$$\mathcal{L}(\text{ł}, \acute{\text{P}}, Q) = \mathcal{L}(\text{ł}) + \mathcal{L}(\acute{\text{P}}) + \mathcal{L}(Q) \tag{6}$$

The propagation delay is calculated by dividing the distance by the propagation speed, which is a fixed value for each physical link. The propagation delay is calculated for all physical links allocated to the virtual links as follows:

$$\mathcal{L}(\text{ł}) = \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\left(D^{e}_{\text{ł}}.y^{e}_{sl}\right) \tag{7}$$

The processing delay refers to the time of executing VNFs in physical nodes, and can be calculated by dividing the total processing demands by the node's processing capacity. The virtualization delay of a service chain depends on the number and load of VNFs placed on a physical node. Eq. (8) formulates the processing and virtualization delay for all physical nodes that are hosting VNFs of a service chain. We model both processing and queuing delays based on M/M/1 queuing models.

$$\mathcal{L}(\acute{\text{P}}) = \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}\frac{1}{(cap^{v}_{cpu} - \sum_{f\in F} d^{v}_{cf}X^{v}_{sf})} + \sum_{s\in S}\sum_{v\in V}\sum_{f\in F} D^{v}_{vir}.X^{v}_{sf} \tag{8}$$

The queuing delay experienced by a service chain depends on the link utilization and can be calculated by dividing the total traffic demand by the link capacity of all nodes that are hosting VNFs consisting of intermediate nodes.

$$\mathcal{L}(Q) = \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\frac{1}{(cap^{e}_{bw} - \sum_{l\in L} d^{e}_{bwl}y^{e}_{sl})} \tag{9}$$

Combining Eq. (7–9), the end-to-end latency model for all service chains is formulated in Eq. (10).

$$\begin{aligned}\mathcal{L}(\text{ł}, \acute{\text{P}}, Q) = &\sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\left(D^{e}_{\text{ł}}.y^{e}_{sl}\right) \\ &+ \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}\frac{1}{(cap^{v}_{cpu} - \sum_{f\in F} d^{v}_{cf}X^{v}_{sf})} + \sum_{s\in S}\sum_{v\in V}\sum_{f\in F} D^{v}_{vir}.X^{v}_{sf} \\ &+ \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\frac{1}{(cap^{e}_{bw} - \sum_{l\in L} d^{e}_{bwl}y^{e}_{sl})}\end{aligned} \tag{10}$$

3.3 Joint cost and latency optimization

Both deployment cost of service chains and end-to-end latency are important objectives in SFC placement. However, they are conflicting objectives; that is, minimizing one could increase the other. The reason is that minimizing the number of active physical nodes increases the network traffic and link congestion and consequently may increase the queuing, processing and virtualization delay. On the other hand, increasing number of physical nodes increases overall costs and even propagation delay (in the case of using an inefficient placement), however, decreases the queuing, processing and virtualization delay. Hence, a proper VNF placement and link embedding is necessary to proper deployment of service chain in edge cloud systems.

Eq. (11) combines cost and latency models using a weight factor $\alpha \in [0, 1]$ as follows:

$$O\left(\text{Ç}, \mathcal{L}\right) = \alpha\text{Ç}(\text{ŗ}, \text{ŋ}, \text{ŧ}) + (1-\alpha)\mathcal{L}(\text{ł}, \acute{\text{P}}, Q) \tag{11}$$

We can adjust α to obtain any needed cost/latency tradeoff, since service providers may have various types of desires and requirements. To achieve more optimal placement according to cost objective, α should be set to a larger number so that $\alpha = 1$ only consider cost optimization and neglect the impact of latency. On the other hand, the smaller α emphasizes the latency minimization so that $\alpha = 0$ optimizes the system only based on latency.

The problem of SFC placement to joint deployment cost and end-to-end latency minimization is formulated as follows:

minimize

$$
\begin{aligned}
O(\zeta, L) = \alpha\Big(& \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}(co_{sf}^{v_{cpu}} + co_{sf}^{v_{mem}} + co_{sf}^{v_{st}}).X_{sf}^{v} \\
& + \Big(\Upsilon.\sum_{v\in V} z^{v} + \varphi.\sum_{v\in V}\Big(\mathrm{p}_{v}^{min} \\
& + (\mathrm{p}_{v}^{max} - \mathrm{p}_{v}^{min})\frac{1}{Cap_{cpu}^{v}}\sum_{f\in F} d_{cf}^{v}X_{f}^{v}\Big).z^{v} \\
& + \sum_{k\in K}\sum_{v\in V}\sum_{f\in F}\mathrm{I}_{k}.X_{kf}^{v}\Big) \\
& + \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\big(co_{sl}^{e_{bw}}.y_{sl}^{e}\big)\Big) \\
& + (1-\alpha)\Big(\sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\big(D_{\mathrm{l}}^{e}.y_{sl}^{e}\big) \\
& + \sum_{s\in S}\sum_{v\in V}\sum_{f\in F}\frac{1}{(cap_{cpu}^{v} - \sum_{f\in F} d_{cf}^{v}X_{sf}^{v})} + \sum_{s\in S}\sum_{v\in V}\sum_{f\in F} D_{vir}^{v}.X_{sf}^{v} \\
& + \sum_{s\in S}\sum_{e\in E}\sum_{l\in L}\frac{1}{(cap_{bw}^{e} - \sum_{l\in L} d_{bwl}^{e}y_{sl}^{e})}\Big)
\end{aligned}
\tag{12}
$$

subject to constraints (13–26).

3.4 Explanation of the optimization problem constraints

The constraints considered for the optimization problem are illustrated in following parts.

3.4.1 Resource constraints

The processing, memory and storage demands of VNFs mapped on the physcial nodes should not exceed the remained CPU, memory and storage capacity of the relevant nodes to avoid resource overutilization. Therefore, Eq. (13–15) guarantee the CPU, memory and storage utilization constraints, repectiverly.

$$\sum_{v\in V}\sum_{f\in F}\left(d_{cf}^{v}X_{sf}^{v}\right) < Cap_{cpu}^{v}, \forall s\in S \tag{13}$$

$$\sum_{v\in V}\sum_{f\in F}\left(d_{mf}^{v}X_{sf}^{v}\right) < Cap_{mem}^{v}, \forall s\in S \tag{14}$$

$$\sum_{v\in V}\sum_{f\in F}\left(d_{stf}^{v}X_{sf}^{v}\right) < Cap_{st}^{v}, \forall s\in S \tag{15}$$

Likewise, for every service chain, the bandwidth allocated to a virtual link $l \in L$ mapped on a physical link $e \in E$ should not exceed the bandwidth capacity of the link for all links in the physical path. Eq. (16) ensures the link's bandwidth usage constraint.

$$\sum_{e\in E}\sum_{l\in L}\left(d_{bwl}^{e}.y_{l}^{e}\right) < cap_{bw}^{e} \tag{16}$$

3.4.2 Placement constraints

The Eq. (17) enforces that all VNFs of a specific service chain must be placed on the available physical nodes working together based on a pre-defined order. Nevertheless, each VNF instance of a given service chain can be placed and instantiated on only one physical node which is guaranteed by the constraint defined in Eq. (18).

$$\sum_{v\in V}\sum_{f\in F} X_{sf}^{v} = |F| \quad , \forall s\in S \tag{17}$$

$$\sum_{v\in V} X_{sf}^{v} = 1 \qquad , \forall f\in F, \forall s\in S \tag{18}$$

Similarly, for every service chain, all the virtual links between each pair of VNFs must be embedded on physical links, as shown in Eq. (19).

$$\sum_{e\in E}\sum_{l\in L} y_{sl}^{e} = |L|, \forall s\in S \tag{19}$$

The constraint defined in Eq. (20) ensures that all VNFs of each service chain should be placed and processed in a predefined order. It is worth to mention that if X_{sp}^{u} =1 and X_{sq}^{v} =1 and Z_{s}^{pq} =1, then we can summarize w_{spq}^{uv} =1, otherwise it equals to 0.

$$\sum_{u,v\in V} w_{spq}^{uv} = Z_s^{pq}, \forall p,q\in F, s\in S \tag{20}$$

Another constraint defined in constraint (21) guarantees that for a service chain which is mapped to a physical path, all its VNFs must be mapped to the nodes belonging to that given path.

$$\sum_{f\in F}\sum_{v\in V} d_s^{\mathfrak{p}} X_{sf}^{\mathfrak{p}v} = m \quad ,\forall\, s\in S, \mathfrak{p}\in P \tag{21}$$

In the case that source and destination of a service demand are the same node, the constraint formulated in Eq. (22) ensures that no physical link will be assigned. Given a service chain s, if a virtual link l between virtual nodes p and q, is allocated on physical link e between physical nodes u and v, then $y_{sl}^{e} = 1$ and consequently $y_{l_{pq}}^{e_{uv}}$.

$$\sum_{e_{uv}\in E} y_{l_{pq}}^{e_{uv}} \le 2 - X_p^v - X_q^v, \forall p,q\in F, \forall u,v\in V, \forall l\in L \tag{22}$$

The constraint defined in Eq. (23) ensures that if there is affinity rule between two VNFs, they will be assigned to a same physical node.

$$X_{sp}^v X_{sq}^v \ge A_{pq}, \forall s\in S, \forall v\in V, \forall p,q\in F \tag{23}$$

Unlike the prior constraint, the constraint defined in Eq. (24) ensures that if there is an anti-affinity rule between two VNFs, they will not be assigned to a same physical node.

$$X_{sp}^v X_{sq}^v AA_{pq} = 0, \forall s\in S, \forall v\in V, \forall p,q\in F \tag{24}$$

3.4.3 Flow constraint

Eq. (25) enforces flow conservation constraint, which means the incoming flow must be equal to the outgoing flow for all nodes.

$$\sum_{e_{uv}\in E} y_{l_{pq}}^{e_{uv}} - \sum_{e_{vu}\in E} y_{l_{pq}}^{e_{vu}} = X_p^v - X_q^v, \forall p,q\in F, \forall u,v\in V, \forall l\in L \tag{25}$$

3.4.4 Latency constraint

Eq. (26) ensures that the end-to-end latency experienced by each service chain flow must not exceed the maximum latency tolerance for the given service chain. Although one of two objective functions in our optimization model is minimizing end-to-end latency, still we need this constraint to guarantee that regardless of the amount of weight factor α, all service demands will be served within the expected time considered in SLA between service providers and users. The end-to-end latency is already formulated in the prior sections.

$$D_{end}^s \le D_{Max}, \forall s\in S \tag{26}$$

4 Heuristic-based SFC placement algorithm

The best method to find the optimal placement of service chains is investigating all the physical nodes and links available in the substrate network. However, because this method requires huge amount of time and computational resources in large-scale networks, we propose two heuristic-based algorithms to find a near-optimal solutions for the problem of SFC placement.

4.1 Bee Colony-based heuristic algorithm

Artificial Bee colony mimics the behavior of honey bee colonies, and is an efficient metaheuristic to solve the NP-hard combinational problems [32]. We solve our proposed optimization problem using an improved bee colony algorithm (BCHA). The proposed algorithm includes three types of bees: employed, onlookers and scout bees. The number of food sources is equal to the number of employed bees in a hive (population). Each location of food sources indicates a potential solution for the optimization problem, and the nectar amount represents the fitness value of each solution. The propose algorithm is shown in Algorithm 1. BCHA includes the following phases:

4.1.1 Initialization

As the first phase, BCHA generates the initial population. The size of population is equal to the total number of food sources (solutions) denoted by P which is defined as the half of the colony size denoted by $cSize$ ($P = cSize/2$). Every solution consists of N nectars (places). Using the following equation, the solutions of the initial population are generated:

$$x_{ij} = lb + R.(ub - lb) \tag{27}$$

Algorithm 1: BCHA

Input: Set of service chains S, set of physical nodes V, set of physical links E,
Output: Embed all service chains in S to physical nodes in V and physical Links in E

1: Set limit , population size P and weight factor $\propto$
2: **for** i=1 to P **do**
3: generate food source f_i using Eq. (27)
4: **if** f_i ! $feasible$ **then**
5: **Go to** the line 3
6: **else**
7: population $pop_i \leftarrow f_i$
8: **end**
9: **end**
10: **for** i=1 to P **do**
11: calculate the normalized cost fitness values
$$NC(f_i) = \frac{C(f_i) - C(f_{min}) \times}{C(f_{max}) - C(f_{min})}$$
12: calculate the normalized latency fitness value
$$NL(f_i) = \frac{L(f_i) - L(f_{min})}{L(f_{max}) - L(f_{min})}$$
13: calculate fitness value $F(f_i) = \propto . NC(f_i) + (1 - \propto) . NL(f_i)$
14: **end**
15 sort pop
16: $dv_1, dv_P \leftarrow \infty$
17: **for** i=2 **to** P-1 **do**
18: $dv_i = F(f_{i+1}) - F(f_{i-1})$
19: **end**
20: $dv_i \leftarrow$ find the largest finite dv as the elite
21: **for** i=1 **to** P **do** // send employed bees
22: modify f_i using Eq. (28)
23: **if** f_i ! $feasible$ **then**
24: go to the line 21
25: **else if** $F(f_i(new)) \leq F(f_i(old))$
26: $pop_i \leftarrow f_i$
27: **else**
28: T_i +=1;
29: **end**
30: **end**
31: assign a selection probability to each f_i using $pr_i = \frac{F(f_i)}{\sum_{i=1}^{P} F(f_i)}$
32: convert pr_i using the following equation and sort pop
$pr_i = (1 - 10)(\frac{F(f_i) - F(f_{min})}{F(f_{max}) - F(f_{min})}) + 10$
33: **for** i=1 **to** P **do** // send onlooker bees
34: select f_i using roulette wheel technique
35: do the lines 21 to 28
36: **end**
37: select the f_i with maximum T // send the scout bees
38: **if** T_i> limit **then**
39: create a new food source f_i using Eq. (27)
40: **if** f_i! $feasible$ **then**
41: go to the line 38
42: **else**
43: $T_i = 0$
44: $pop_i \leftarrow f_i$
45: **end**
46: **end**
47: **if** the convergence condition is true **then**
48: return the solution
49: **else** continue to the next iteration
50: **end**

Where i=1, 2, …, P; j = 1, 2, …, N; x_{ij} represents each nectar in the solution, lb denotes the lower bound (in our work it equals to zero), R is a random number within interval [0,1] and ub is the upper bound (in our work it equals to the maximum number of physical nodes minus 1).

4.1.2 Send employed bees

In this phase, for each food source we send an employed bee to probe a new food source which has a better amount of nectar. In the other word, each solution in the population will be replaced by a new solution that obtains a better fitness value using Eq. (28).

$$y_{ij} = x_{ij} + R_1.(x_{ij} - x_{kj}) + R_2.(x_{ij} - e_d) \tag{28}$$

Where R_1 and R_2 are two different random numbers over interval [0,1], and k is a random integer in range [0, P-1] and different from i, and e refers to the elite solution. Elite is a solution with highest distance value (dv) [32] and can be calculated as follows:

$$dv_i = F(f_{i+1}) - F(f_{i-1}) \tag{29}$$

Where $F(f_i)$ refers the fitness value of the solution i. We prefer a solution with higher dv since it reveals that the solution is located in a less crowded region. Therefore, it helps the algorithm to explore a wider area and increases the diversity of the solution. The algorithm defines a counter for each solution to keep the number of times that a given solution is not optimized in each stage. In the case that the new solution is not obtain a better fitness rather than the current solution, the counter will be incremented. If the counter of each solution exceeds a limit, the relevant bee is considered as a scout bee. It is worth mentioning that, unlike the original roulette wheel method that gives more probability to the selection of solutions with higher fitness values, the roulette wheel in our proposed methods gives more probability to those with lower fitness values. This is because, in line with the minimization nature of our problem statement, we have deigned our fitness values so that better solutions receive less fitness values. Fitness values are mapped to a real number in the range [1, 10], as shown in line 32 in Algorithm 1.

4.1.3 Send onlooker bees

In the previous phase, all employed bees tried to optimize their food sources. Then, in this phase, employed bees come to hive to advertise their food sources. Each onlooker bee should decide and select a food source to be utilized. In other words, BCHA assigns a probability to every solution in population by which the solutions with lower fitness values get higher selection probability since our target is minimizing cost and latency in this work. For all the places in the population, the algorithm selects a solution using roulette wheel selection technique and tries to optimize it using Eq. (28), the same as in the previous phase.

Once again, in the case that the new solution cannot be further improved, the relevant counter will be incremented

and if the pre-defined limit is violated, the solution will be considered as a scout bee.

4.1.4 Send scout bees

In this phase, the scout bees (abounded solutions) are found and replaced with a new solution which is generated using Eq. (27). In each iteration only one scout bee will be replaced. The abandoned solutions should be removed because they have been trapped in a local optimum. Then, the relevant counter for new solutions turn to zero again. Then, the algorithm will be repeated until the convergence condition is reached.

4.2 GA-based heuristic algorithm

As an evolutionary algorithm, genetic algorithm imitates the natural evolution so that the solution proceeds towards a more optimal solution after each generation. We propose a modified GA-based algorithm (GAHA) designed to solve the SFC placement problem. The proposed algorithm is shown in Algorithm 2. GAHA includes several phases as follows:

4.2.1 Encoding scheme

In the proposed algorithm, each potential placement solution is defined as a chromosome (individual). Every individual consists of a number of genes represented by an integer [1, N] interval, where N is the number of physical nodes, and each gene corresponds to a VNF mapped to a physical node. Individuals are initiated randomly and must satisfy the constraints of the optimization model.

4.2.2 GA operators

Genetic algorithm aims to merge the individuals to create offspring using crossover operator and then mutate them using mutation operator. Each offspring inherits its genes from two parents selected for crossover. First, all the individuals in the population are sorted in an increasing order in terms of their fitness values. The fitness function is defined based on the objective function formulated in Eq. (12), and uses normalized cost and latency values. Next, a number of individuals from the top of the list with lowest fitness values are selected as elites based on the elitism rate, and they will be directly added to the new population. Given the crossover rate, the offspring are created from the parents which are selected using the roulette wheel selection technique.

The proposed algorithm is a self-healing algorithm; that is, after generating each offspring all the constraints will be checked and must be satisfied, otherwise for the same parents, crossover operator will generate another offspring which satisfies the constraints. To avoid getting stuck in an infinite loop and consequently increasing computational time, we set a condition using a constant (β). It determines maximum number of times that algorithm tries to generate a chromosome satisfying the constraint, otherwise, one of the parents will be selected to add to the new population. Crossover operation will be repeated until all offspring individuals have been generated and added to the new population.

Having the new population, mutation as the second GA operator aims to mutate each individual from top of the list. This operator picks one gene randomly and replace it with another gene. New individuals will be checked for feasibility, otherwise the mutation will be

Table 4 Input values

Servers	Num. CPU cores	[16–32]
	Memory capacity (GB)	[4–8]
	Storage capacity (GB)	[500–1000]
	Physical link capacity (Gbps)	10
	Min. power consumption (W)	240
	Max. power consumption (W)	400
	The monetary weight φ	0.1
	Link bandwidth cost per Gbps ($)	10
	CPU cost per core, memory cost per 1GB, and storage cost per 100 GB ($)	[5–10]
	Software license cost is fixed to ($)	1000
SFC	Delay Threshold (ms)	[50–100]
	Num. of VNFs in SFCs	[4–10]
VNF	CPU demand	[1–8]
	Memory demand (MB)	[100–400]
	Storage demand	[10–50]

Algorithm 2: GAHA

Input: Set of service chains S, set of physical nodes V, set of physical links E,
Output: Embed all service chains in S to physical nodes in V and physical Links in E

1: Set elite rate $\grave{e}$, population size $pSize$ and weight factor $\propto$
2: **for** i=1 to $pSize$ **do**
3: generate distributed randomly chromosomes c_i
4: **if** c_i ! $feasible$ **then**
5: **Go to** the line 3
6: **else**
7: population $pop_i \longleftarrow c_i$
8: **end**
9: **end**
10: **for** i=1 to $pSize$ **do**
11: Calculate the normalized cost fitness values
$$NC(c_i) = \frac{\mathrm{C}(c_i) - \mathrm{C}(c_{min})}{\mathrm{C}(c_{max}) - \mathrm{C}(c_{min})}$$
12: Calculate the normalized latency fitness value
$$NL(c_i) = \frac{L(c_i) - L(c_{min})}{L(c_{max}) - L(c_{min})}$$
13: Calculate fitness value $\mathrm{F}(c_i) = \propto . NC(c_i) + (1-\propto) . NL(c_i)$
14: **end**
15: sort pop
16: **for** i=1 **to** $(\grave{e} \times pSize), E$ **do**
17: $neWpop_i = elite_i$
18: **end**
19: assign a selection probability to each c_i using $pr_i = \frac{F(c_i)}{\sum_{i=1}^{pSize} F(c_i)}$
20: convert pr_i using the following equation and sort pop
$pr_i = (1-10)(\frac{F(f_i)-F(f_{min})}{F(f_{max})-F(f_{min})}) + 10$
21: **for** E+1 **to** $pSize$ **do** // crossover
22: select parents p and q using roulette wheel technique
23: **for** j=1 **to** $chrmosome\ size, cSize$ **do**
24: randomly generate $r \in [0,1]$
25: **if** $r \leq F(q)/\,(F(p) + F(q))$
26: offspring $o_{ij} \leftarrow e_i^q$
27: **else**
28: $o_{ij} \leftarrow e_{ij}^p$
29: **end**
30: **end**
31: **if** o_i ! $feasible$ **then**
32: **if** repeat $\leq \beta$ **then**
33: go to the line 22
34: **else**
35: $neWpop_i \leftarrow the\ parent\ with\ bigger\ F(c)$
36 **end**
37: **else**
38: $neWpop_i \leftarrow o_i$
39: **end**
40: **end**
41: **for** E+1 **to** $pSize$ **do** // mutation
42: generate a random number in range of 1 to $cSize$, k
43: generate a random number in range 1 to the number of physical nodes, $|V|$
44: $neWpop_{ik} \leftarrow |V|$
45: check the feasibility as mentioned in crossover
46: **end**
47: **if** convergence condition is true **then**
48: return the solution
49: **else** continue to the next generation
50: **end**

repeated until (β) times. If the individual still is not valid, the prior individual (before mutation) will be considered. Now, the fittest individual will be selected as the solution for that given generation. The GA algorithm continues to generate new populations until converging to the final solution (when the last k generations obtain the same results).

5 Performance evaluation

In this section, we evaluate the efficiency of our optimization model solved by the proposed algorithms through both simulation and Mininet [33]. We start by explaining the experimental settings, continue by presenting the simulation results, and then we demonstrate the measurement results obtained from Mininet. We, also, compare the performance of our work with the optimal solution derived from Gurobi solver [34], and a greedy algorithm named modified first-fit decreasing (FFD).

5.1 Simulation setup

In our work, we used the Geant network topology including 22 nodes and 36 links extracted from SndLib [35]. SndLib's topologies are widely used in literature as substrate networks to simulate the placement algorithms (e.g., see [36, 37]).

In the simulation, each service chain assumes as a virtual network that includes source-destination pairs of virtual nodes (VNs) connected with virtual links (VLs). The set of service chains are taken as input by the optimization program. The proposed metaheuristic algorithms try to find the optimal placement for both VNs and VLs at the same stage with respect to the optimization model. First, for each service chain, they select proper physical nodes to be assigned to VNs, and then relevant VLs are embedded to the physical paths between selected pairs of physical nodes using a shortest path algorithm (Dijkstra algorithm), in each iteration of the algorithm. Each physical node can be potentially selected as a source, destination or intermediate node of one or several VLs. The simulation is run on a single machine with an Intel core i5 CPU@1.8 GHz, and 8 GB of RAM.

We run each algorithm 10 times and the best value is shown in the measurements. Table 4 shows the input values for the servers, SFCs and VNFs used in the experiments.

5.2 Results and analysis

In this sub-section, in order to evaluate the efficiency of the proposed optimization model and algorithms, we compare our works and following approaches and baselines:

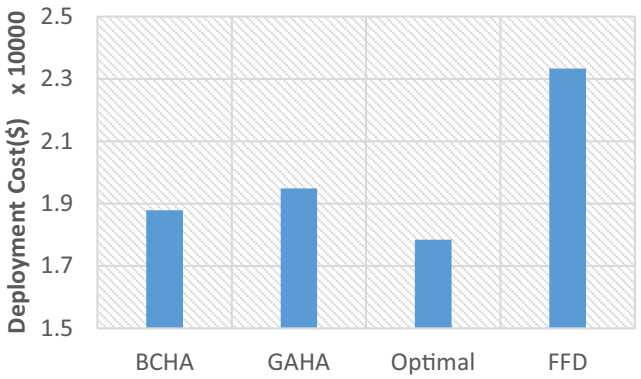

Fig. 2 The proposed algorithms vs optimal solution and FFD in terms of deployment cost

- **BCHA**: The proposed bee colony-based algorithm aims to solve the optimization problem in a way that joint cost and latency are minimized.
- **GAHA**: The proposed GA-based algorithm considers both cost and latency minimization as well.
- **Optimal**: This baseline indicates the exact solution obtained by Gurobi. We aim to compare the accuracy of our solutions rather than optimal solution. It is worth mentioning that the time complexity of optimal solution is exponential in sizable problem, due to innate complexity of SFC placement.
- **FFD**: This algorithm, first, sorts all physical nodes in decreasing order regarding their resource capacities, and then allocates maximum number of VNFs to the nodes from top of the list with respect to the capacity, affinity and anti-affinity constraints. We aim to evaluate the efficiency of our optimization solutions compared with a non-optimized greedy approach.

5.2.1 Cost quality of solution

As the first metric, we evaluate the performance of the proposed algorithms with optimal solution FFD algorithm in terms of monetary cost. As we can see in Fig. 2, both proposed algorithms significantly decrease the monetary cost compared with FFD. This amount is around 20% for BCHA. BCHA even obtains the lower amount rather than GAHA, however, the gap between the optimal solution and BCHA is as small as 5%, while execution time is significantly less. It shows that the proposed heuristic based placement algorithms are quite successful in reducing monetary cost.

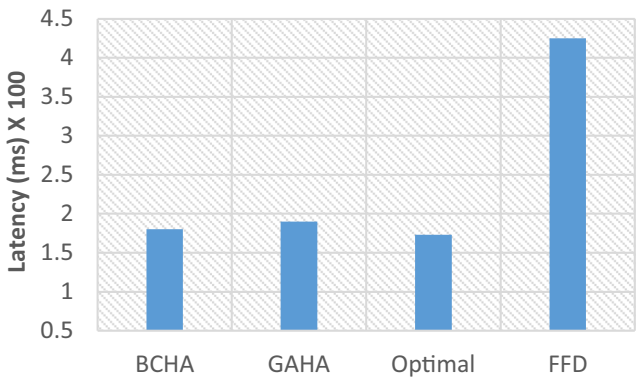

Fig. 3 The proposed algorithms vs optimal solution and FFD in terms of latency

5.2.2 Latency quality of solution

The latency obtained by the proposed algorithms, optimal solution and FFD are compared in Fig. 3. As we can see, BCHA obtains a better result compared to GAHA although the difference is not remarkable, only about 6%. BCHA achieves an end-to-end latency which is about only 4% higher than optimality, but around 57% less than FFD.

Therefore, we can sum up both algorithms improve monetary cost and latency significantly compared with FFD algorithm, and their obtained results are very close to optimal solution which shows the efficiency of the proposed placement algorithms. BCHA reduces both cost and latency more than GAHA although the differences are not very remarkable.

5.2.3 Pareto quality of solution

Figure 4 shows the deployment cost for both BCHA and GAHA for different α values between 0 to 1. As we can see, in general, both algorithms obtained close results in all values, but BCHA is still better in all cases not more than 7% happened in $\alpha = 0.1$. The amount of cost decreases by increasing the amount of α. That is because α is a weight factor which emphasizes on cost optimality rather than latency, so having the higher values of α, optimization model tries to find better cost placement solution and we expect to have lower amount of cost. Therefore, $\alpha = 1$ obtains the lowest amount of

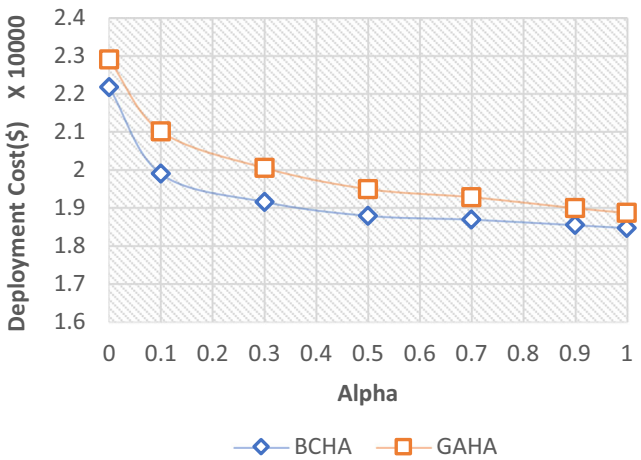

Fig. 4 BCHA vs GAHA for different α values in terms of deployment cost

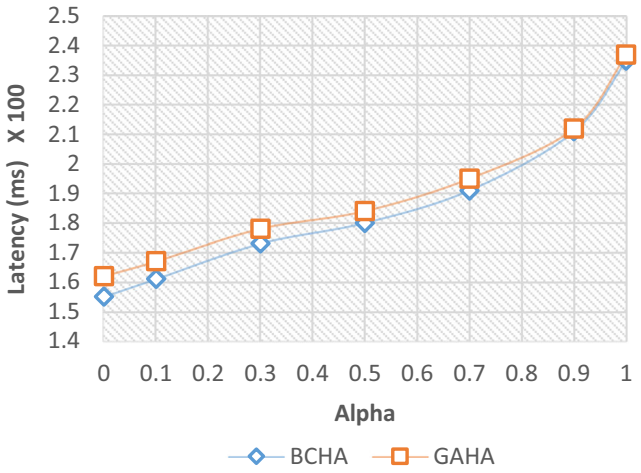

Fig. 5 BCHA vs GAHA for different α values in terms of latency

monetary cost and $\alpha = 1$ obtains the worst cost optimality. From $\alpha = 0$ to $\alpha = 0.5$, cost decreases sharply, but it decreases from $\alpha = 0.5$ to $\alpha = 1$ with a gentle slope.

Figure 5 shows the results obtained by BCHA and GAHA in terms of latency considering different values of α. Unlike what we said in previous metric, higher values of α will bring lower amount of latency; that is, $\alpha = 0$ results in lowest latency amount while $\alpha = 1$ achieves the highest end-to-end latency.

Again, we can see the amount of latency obtained by both algorithms are very close to each other in different values of α, not more than 5%, although BCHA is still more successful in reducing latency. Unlike the cost, from $\alpha = 0.5$ to $\alpha = 1$, latency increases sharply, and from $\alpha = 0$ to $\alpha = 0.5$ it increases with a gentle slope; however, the latency curve is increasing sharper than decreasing in the cost curve in general.

We can observe the pareto optimal results for both latency and cost optimization in Fig. 6. We use normalized cost and latency values, to see BCHA pareto results for different α values. As we can see, by increasing α we can have more optimal placement in terms of cost, however less optimal solution in latency. On the other side, by decreasing α we can outperform latency although the deployment cost increases. However, based on the results, $\alpha = 0.3$ provides the best placement in which joint cost and latency are minimized. Therefore, we can sum up that $\alpha = 0.3$ achieves a balanced optimality between both objectives, although α can be selected based on the requirements of services and desires of service providers.

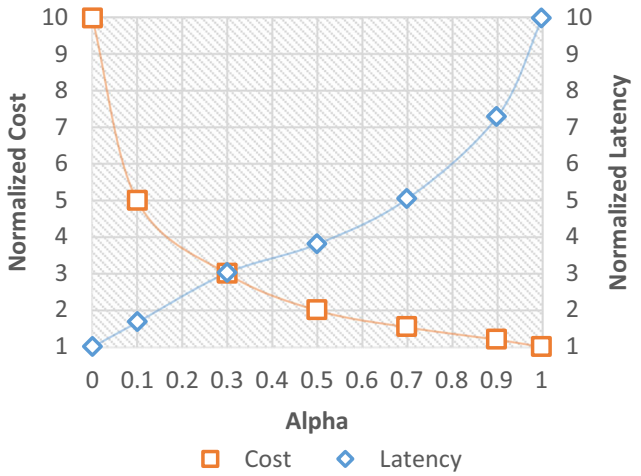

Fig. 6 Pareto optimal results obtained by BCHA for different values of α for both cost and latency

5.2.4 Accuracy of solution

In this subsection, our goal is to determine whether the results obtained via simulations can qualitatively match the results measured in Mininet. In this experiment, we use Dijkstra algorithm to find the shortest path between physical nodes that host the VNFs for each service chain, as in the simulations. We developed a Ryu controller in Mininet which is able to place and proceed the SFCs on physical nodes, then we obtained the end-to-end latency for SFCs based on the placement solutions determined via simulation. In Fig. 7 we evaluate the latency results obtained by simulation compared with Mininet for BCHA.

As it can be observed, for different values of VNFs, the latency achieved by emulation using Mininet is very close to the simulation results not more than 6% in all cases. It proves the validity of simulation results in terms of end-to-end latency of service chains.

5.2.5 Impact of service chains complexity

In this subsection, we investigate the impact of service chain complexity on the cost and latency of the system. The complexity of an application is determined by the length of service chain or the number of VNFs in the service chain. Table 5

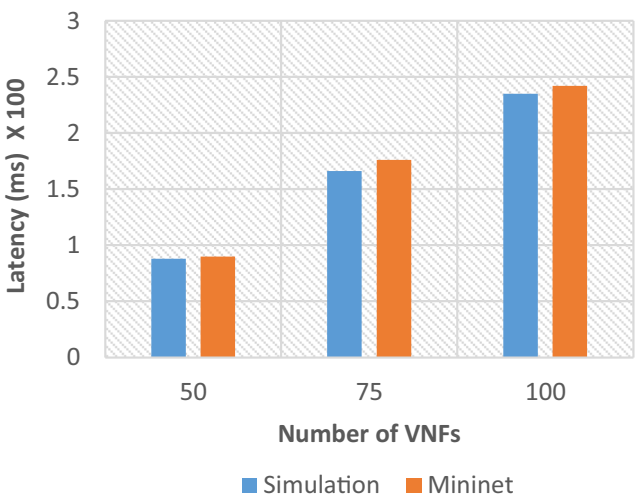

Fig. 7 Latency results obtained by BCHA using simulation vs Mininet

Table 5 Cost and latency obtained by BCHA approach using different number of VNFs

Number of VNFs in each SFC	Cost ($)	Latency (*ms*)
4	1.26×10^5	1.33×10^2
5	1.58×10^5	1.69×10^2
6	1.89×10^5	1.91×10^2
7	2.22×10^5	2.25×10^2

shows the cost and latency results obtained by BCHA approach for the service chains formed by the fixed number of VNFs as: 4,5,6 and 7 VNFs. From Table 5, we can find that both cost and latency of SFCs deployment increase when the application become complex. The reason is clear because the more VNFs to be deployed, the more instances, resources, power and physical nodes should be used, and consequently the monetary cost will be increased. At the same time, increasing in the number of virtual links connecting VNFs will increment end-to-end latency as well.

6 Conclusion

In summary, we investigated the problem of SFCs placement in edge/cloud environments. Service providers tend to minimize the deployment cost of service chains. However, it may increase the end-to-end latency of services which increases SLA violation and decreases user satisfaction. In this paper, we proposed a multi-objective and comprehensive optimization model to joint minimizing deployment cost and end-to-end latency considering a variety of parameters. We, then, proposed two heuristic-based placement algorithms using genetic algorithm (GAHA) and bee colony algorithm (BCHA).

We evaluated the efficiency of our approach using both simulation and emulation (Mininet). The proposed algorithms are compared to optimal solution obtained by Gurobi as well as the FFD placement algorithm. The results showed that BCHA outperforms monetary cost rather than GAHA by 4% and FFD by 20% while its results are close to optimal solution (5%). In terms of latency, BCHA reduces the end-to-end latency compared to GAHA by 6% and FFD by 57% while is about only 4% higher than optimality. Therefore, we can sum up the proposed algorithms are quite efficient in joint cost and latency optimization. In addition, we proposed the pareto optimal solutions for both cost and latency objectives for different values of α which can be used to make a balance between conflicting objectives based on the desired goals. Finally, we emulated our work using Mininet to validate the latency results obtained through simulation. We observed that the Mininet results are only about 6% higher than the result obtained through simulation. As a future work, we plan to take into account the network traffic as another important objective in the optimization model along with VNF sharing policy in the SFCs placement.

Acknowledgments This paper was partially supported by the Grant No. 20160182 funded by the Knowledge Foundation of Sweden, and also supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61902236.

Funding Open access funding provided by Karlstad University.

References

1. Zhang W, Liu G, Zhang W, Shah N, Lopreiato P, Todeschi G, Wood T (2016) OpenNetVM: a platform for high performance network service chains. In proceedings of the 2016 workshop on hot topics in Middleboxes and network function virtualization: 26-31
2. Li Y, Chen M (2015) Software-defined network function virtualization: a survey. IEEE Access 3:2542–2553
3. Pham C, Tran NH, Ren S, Saad W, Hong CS (2017) Traffic-aware and energy-efficient vnf placement for service chaining: joint sampling and matching approach. IEEE Trans Serv Comput
4. Bari MF, Chowdhury SR, Ahmed R, Boutaba R (2015) On orchestrating virtual network functions. In Proc. 11th Int. Conf. Netw. Service manage. (CNSM): 50–56
5. Luizelli MC, Bays LR, Buriol LS, Barcellos MP, Gaspary LP (2015) Piecing together the NFV provisioning puzzle: efficient placement and chaining of virtual network functions. In Proc IFIP/IEEE Int Symp Integr Netw Manag: 98–106
6. Amaldi E, Coniglio S, Koster AM, Tieves M (2016) On the computational complexity of the virtual network embedding problem. Electron Notes Discrete Math 52:213–220
7. Kuo TW, Liou BH, Lin KCJ, Tsai MJ (2016) Deploying chains of virtual network functions: on the relation between link and server usage. In Proc. IEEE INFOCOM, 35th Annu. IEEE Int. Conf. Comput Commun 1–9
8. Mijumbi R, Serrat J, Gorricho JL, Bouten N, De Turck F, Davy S, Design and evaluation of algorithms for mapping and scheduling of virtual network functions. in Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft): 1–9
9. Agarwal S, Malandrino F, Chiasserini CF, De S (2018) Joint VNF Placement and CPU Allocation in 5G. In IEEE INFOCOM 2018-IEEE Conference on Computer Communications:1943–1951
10. Gao H, Huang W, Duan Y(2020). The cloud-edge based dynamic reconfiguration to service workflow for Mobile ecommerce environments: a QoS prediction perspective. ACM Trans Internet Technol

11. Gao H, Xu Y, Yin Y, Zhang W, Li R, Wang X (2019) Context-aware QoS prediction with neural collaborative filtering for internet-of-things services. IEEE Internet Things J
12. Yin Y, Chen L, Xu Y, Wan J, Zhang H, Mai Z (2019) QoS prediction for service recommendation with deep feature learning in edge computing environment. Mobile Netw Appl
13. Bruschi R, Carrega A, Davoli F (2016) A game for energy-aware allocation of virtualized network functions. J Elect Comput Eng 2016(2):4067186
14. Taleb T, Ksentini A, Sericola B (2016) On service resilience in cloudnative 5G mobile systems. IEEE J Sel Areas Commun 34(3):483–496
15. Beck MT, Botero JF, Samelin K (2016) Resilient allocation of service function chains. In Proc. IEEE Conf. Netw. Function virtualization Softw. Defined Netw. (NFV-SDN):128–133
16. Bhamare D, Kassler A, Vestin J, Khoshkholghi MA, Taheri J (2019) IntOpt: in-band network telemetry Optimization for NFV service chain monitoring. In IEEE international conference on communications (ICC), (pp. 1-7)
17. Kuang L, Gong T, OuYang S, Gao H, Deng S (2020). Offloading decision methods for multiple users with structured tasks in edge computing for smart cities. Future generation computer systems(FGCS)
18. Mehraghdam S, Keller M, Karl H (2014) Specifying and placing chains of virtual network functions. In Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet): 7–13
19. Chen Y, Deng S, Ma H, Yin J (2020) Deploying Data-intensive Applications with Multiple Services Components on Edge. Monet 25(2):426–441
20. Deng S, Xiang Z, Zhao P, Taheri J, Gao H, Yin J, Zomaya A (2020). Dynamical resource allocation in edge for trustable iot systems: a reinforcement learning method. IEEE Trans Ind Inf
21. Khoshkholghi MA, Taheri J, Bhamare D, Kassler A, (2019). Optimized service chain placement using genetic algorithm. In IEEE conference on network Softwarization (NetSoft): 472-479
22. Rankothge W, Le F, Russo A, Lobo J (2017) Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms. IEEE trans. On. Network Service Manag (TNSM) 14(2):343–356
23. Xia M, Shirazipour M, Zhang Y, Green H, Takacs A (2015) Network function placement for NFV chaining in packet/optical datacenters. J Lightwave Technol 33(8):1565–1570
24. Cohen R, Lewin-Eytan L, Naor JS, Raz D (2015) Near optimal placement of virtual network functions. In computer communications (INFOCOM), 2015 IEEE conference on:1346-1354
25. Mijumbi R, Serrat J, Gorricho J, Bouten N, Turck FD, Davy S (2015) Design and evaluation of algorithms for mapping and scheduling of virtual network functions, in proceedings of the 1st IEEE conference on network Softwarization. NetSoft 2015:1–9
26. Wang S, Urgaonkar R, Zafer M, He T, Chan K, Leung K. (2015) Dynamic service migration in mobile edge-clouds. In IFIP Networking Conference:1–9
27. Fan Q, Ansari N, (2017) Cost aware cloudlet placement for big data processing at the edge. In proceedings of the IEEE international conference on communications (ICC 2017): 1–6
28. Chua FC, Ward J, Zhang Y, Sharma P, Huberman BA (2016) Stringer: balancing latency and resource usage in service function chain provisioning. IEEE Internet Comput 20(6):22–31
29. Ahvar S, Sahoo J, Ahvar E, Dieye M, Glitho R, Elbiaze H, Crespi N (2018) PCPV: pattern-based cost-efficient proactive VNF placement and chaining for value-added services in content delivery networks. In 2018 4th IEEE conference on network Softwarization and workshops (NetSoft): 313-317
30. Khoshkholghi MA, Derahman MN, Abdullah A, Subramaniam S, Othman M (2017). Energy-efficient algorithms for dynamic virtual machine consolidation in cloud data centers. IEEE Access 2017;5: 10709–10722
31. Khoshkholghi MA, Abdullah A, Subramaniam S, Othman M, Derahman MN (2016) A taxonomy and survey of power management strategies in cloud data centers. Int J Commun Antenna Propag 6(5):305–317
32. Xiang Y, Zhou Y, Liu H (2015) An elitism based multi-objective artificial bee colony algorithm. Eur J Oper Res 245(1):168–193
33. "Mininet" [Online]. Available: http://mininet.org. Accessed May-2020
34. Gurobi Optimization. [Online]. Available: http://www.gurobi.com. Accessed May-2020
35. "SNDlib." [Online]. Available: http://sndlib.zib.de. Accessed May-2020
36. Raayatpanah MA, Weise T. (2018) Virtual network function placement for service function chaining with minimum energy consumption. In IEEE international conference on computer and communication engineering technology (CCET): 198-202
37. Reddy VS, Baumgartner A, Bauschert T (2016) Robust embedding of VNF/service chains with delay bounds. In 2016 IEEE conference on network function virtualization and software defined networks (NFV-SDN): 93-99