# Learning Framework for IoT Services Chain Implementation in Edge Cloud Platform

Chuan Pham*, Duong Tuan Nguyen*, Nguyen H. Tran†, Kim Khoa Nguyen*, Mohamed Cheriet*,
*Synchromedia - École de Technologie Supérieure, Université du Québec, H3C1K3, Canada,
† School of Computer Science, The University of Sydney, NSW, Australia.

***Abstract*—As an emerging solution to latency requirements of Internet of Things (IoT) services, edge computing can bring powerful processing capacity closer to data sources. However, with the limited resources at edge nodes, a major challenge is finding optimal resources in distributed edges to reduce the operational costs of service deployment. Prior works focus mainly on static optimization which may not work efficiently with the time-varying workloads and resource constraints. In this paper, we, therefore, consider a dynamic allocation framework in the edge-cloud network over the long run with uncertainty workloads. In such a system, we introduce a JOint Routing and Placement problem for IoT services, called JORP, that dynamically assigns resources according to workload demand in order to reduce the operational costs in long term. Inspired from the well-known algorithm, branch-and-bound (BnB), for solving the mixed-integer non linear problems (MINLPs) like JORP, we bring the learning concept to address the high complexity of BnB when the search space is huge. Particularly, we design a deep neural network (DNN) and train it under the imitation learning to mimic branching behaviors in BnB for searching the optimal solution. Finally, simulations show our solution outperforms baselines in terms of convergence and operational cost.**

***Index Terms*—Internet of Things, Edge Computing, Cloud, Resource Allocation, Branch-and-Bound, Deep Neural Network.**

## I. Introduction

With many benefits from virtualization technology, network operators start to adopt this paradigm to transform their hardware infrastructure in both cloud and edge networks. Especially, the edge computing with NFV-based architectures has attracted substantial research attention as an efficient implementation platform for deploying IoT services [1]. In the IoT network, edge nodes [2] are referred to powerful gateways that are used to run client services [2], [3]. Similar to cloud infrastructure where multiple VNFs are hosted, an edge node however takes advantage of the close distance to end nodes and therefore significantly reduces network latency. For example, a temperature service in a smart building places a set of sensor devices in the edge to gather temperature information. There are IoT gateways are responsible for receiving data from sensors then forward to next functions for processing. Such functions can be virtualized and deployed at edge nodes rather than at the cloud in order to meet real-time requirements. Virtualization technologies supporting IoT service deployment in edge networks typically include Docker [4], OpenStack [5], and Kubernetes [6]. Unlike the cloud, an edge node is unable to run too many network functions at the same time due to its limited resource capacity. In a smart city network with

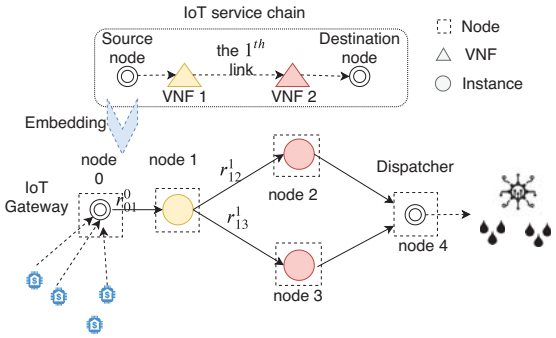



Fig. 1: An example of the VNF routing and placement problem for an IoT service chain.

hundreds of IoT services, a network operator, therefore, needs to carefully utilize limited edge resources and leverage the presence of cloud nodes in an efficient way to reduce the operational costs when deploying IoT services.

In this study, we consider an important use-case of IoT service implementation: dynamically placing IoT services in a smart city where the edge-cloud based architecture [7] is accounted to support this model. In details, to run an IoT service, many VNFs are chained in a specific order (called service chain [8]) and placed in different locations to handle some tasks (e.g., processing temperature data, encoding video streaming). Despite several intensive investigations in literature, such as [8]–[10], resource allocation for deploying IoT service chains is still facing many challenging issues as follows. *First*, the VNF placement problem [8] is often formulated for cloud-based network and a few works consider the system model that fits to IoT networks. In particular, VNFs in the general VNF placement problem [11] can be placed in any available nodes of the substrate network but it does not fit in case of IoT networks. For instance, since some source and destination nodes of an IoT service are related to locations of IoT devices, they are constrained to be placed in some specific nodes. *Second*, because this problem is NP-hard, an efficient solution is still needed. *Lastly*, the VNF placement in IoT networks should be aware of time-varying traffic of the service chain. For instance, the amount of sensing information in a smart building during peak hours increases since many offices and activities are activated; meanwhile in the midnight, most of services only run as background services. This situation makes several instances of VNFs perform with very low utilization. Therefore, designing a dynamic allocation and

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

scaling approach to deploy IoT services in long term is the main objective in our work.

The main contributions of this work are summarized as follows. First, we leverage the flexible capability of NFV in the edge-cloud based architecture to design a dynamic VNF scaling solution for IoT service implementation. Specifically, we take into account resource allocation in the system to optimize the system cost over the long run of the system in which requirements of IoT service constraints, such as the number of VNF instances and the Service Level Agreements (SLA) to implement IoT services are involved. We then formulate the optimization named as JOint dynamic Routing and Placement (JORP) problem for IoT service implementation with the time-varying workload demands. Another challenge is the NP-hardness form of JORP, which is too difficult to find an optimal solution of JORP in polynomial time. In this work, we advocate the branch-and-bound (BnB) solution, a well-known algorithm for solving mixed-integer non linear problem (MINLP) [12], to look for a near-optimal solution of JORP. We also acknowledge the fact that BnB converges very slow due to a large search space and may not fit reality well. This drawback is mitigated by adopting a learning method, named as L-JORP, in which a DNN is designed to learn pruning rules as BnB to search the optimal solution. Furthermore, to "steer" the search into potential branches, we train the model based on the imitation theory, DAgger [13].

## II. Problem Formulation and Solution Framework

### A. System model

In this paper, we consider an enterprise IoT service deployment system, in which a set $\mathcal{N}$ of VNFs is deployed to implement an IoT service chain. The service chain is comprised of VNFs chained in a specific order, each VNF may requires multiple instances to fulfill workload demands. To execute the service chain, VNFs are deployed on a set of physical nodes including edge and cloud nodes. A toy example of an IoT service chain is depicted in Fig. 1 where there are two VNFs in this chain that are placed in the substrate network corresponding to three VNF instances. VNF 1 is placed with only one instance in node 1, and VNF 2 requires two instances placed in different nodes (i.e., node 2 and node 3). The traffic from the source node 0 is steered into two paths to reach the destination, node 4. For ease of notations, we use $n$ to index for the $n^{\text{th}}$ VNF in the chain; especially, the source and destination VNFs are indexed by 0 and $N$, respectively. The $0^{\text{th}}$ VNF is the head of the service chain where the traffic flows are aggregated and passed through the chain. In IoT networks, source nodes are often IoT gateways that are responsible for collecting data from all the sensors. Meanwhile, the destination nodes are played as dispatchers to connect actuators for some specific tasks, such as opening/closing doors, switching lights. Therefore, these functions are fixed in the implementation and they cannot be placed arbitrarily into any physical nodes as many prior works [14]. Given a pair of source and destination nodes, we aim to place VNF instances into nodes and decide a routing allocation in order to utilize resources efficiently while satisfying all the service requirements over long run.

We model our infrastructure as a network graph $(\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ represents the set of nodes and $\mathcal{E}$ is the set of links between them. In particular, two types of nodes are identified as follows: i) server nodes $\mathcal{V}^s \subset \mathcal{V}$ that are used to place VNFs; ii) router or gateway nodes that are used for traffic forwarding and routing. We denote the amount of available computing resources of node $v \in \mathcal{V}$ by $p_v$. Practically, there are multiple resource types on each server, such CPU, memory, storage, network bandwidth, etc; however, we only consider one resource type to simplify the formulation that is readily extensible in the general case. Furthermore, we consider that the system works in a spanning time slots $1, 2, ..., T$. At timeslot $t$, given the fact that the system can receive several requests, we suppose all of requests are organized to a queue and executed according to First-Come, First-Served (FCFS).

### B. Problem formulation

To formulate the joint dynamic VNF placement and routing problem, we define following allocation variables: i) $x_{nv}(t)$, an integer variable, to indicate the number of instances of $n^{\text{th}}$ VNF placed on node $v$ at time $t$; ii) $r^n_{uv}(t)$, an real variable, to indicate the traffic rate allocation of link $n^{\text{th}}$ of the service chain over the physical link $(uv)$.

*1) Service chain constraint:* Since a physical node can place multiple instances of a VNF, a placement solution cannot provide resource over the capacity of a node. Hence the constraint is formulated as follows:

$$\sum\nolimits_{n\in\mathcal{N}} x_{nv}(t)p_n \leq \kappa_v p_v, \forall v \in \mathcal{V}, \tag{1}$$

where $\kappa_v$ is the decay factor of each node $v$ and $p_n$ is the required resources of an instance of the $n^{\text{th}}$ VNF.

Each VNF in the service chain has to be placed at least one instance to execute a specific task as follows:

$$\phi_n(t) = \sum\nolimits_{v\in\mathcal{V}} x_{nv}(t) \geq 1, \forall n \in \mathcal{N}. \tag{2}$$

We also need to ensure that all the requests will be served through every VNF according a specific order of the service chain as follows

$$\sum\nolimits_{v\in\mathcal{V}} r^0_{uv}(t) \geq \lambda(t), t = 1, ..., T, \tag{3}$$

where $\sum_{v\in\mathcal{V}} r^0_{uv}(t)$ represents the aggregation traffic forwarded to instances of the $1^{\text{st}}$ VNF that is placed in node $v$ and $\lambda(t)$ is the given traffic loads that need to be served.

Second, the traffic rate of a service chain departs at the $0^{\text{th}}$ VNF. This rate is changed after being processed by a specific VNF. Hence we denote $\alpha_n$ as a change ratio after processing at the $n^{\text{th}}$ VNF. To conserve all of traffic going through VNFs to avoid the packet loss, we represent an incoming-outgoing traffic at VNF $n$ as follows

$$\alpha_n \sum_{u\in\mathcal{V}^s} r^{n-1}_{uv}(t) = \sum_{v'\in\mathcal{V}^s} r^n_{uv'}(t). \tag{4}$$

where (4) presents the total outgoing traffic at VNF $n$ that must equal the value of the multiplication between the total incoming traffic and the change rate.

*2) Latency constraint:* If the $n^{\text{th}}$ VNF is deployed with many instances, the traffic will be steered equally among instances. Given the input workload $\lambda(t)$ of the service chain

at time $t$, the arrival workload at the $n^{\text{th}}$ VNF can be calculated by the product of the initial rate and all the change rates of its previous VNFs in the service chain, that is

$$\lambda_n(t) = \lambda(t) \prod_{m=1}^{n-1} \alpha_m, t = 1, 2, ..., T. \quad (5)$$

A node has a processing rate $\mu_v$ that is shared between instances of VNFs. The more instances that are placed in a node the higher processing delay the node includes. Accounting the latency at the $n^{\text{th}}$ VNF, we define $L_n^{pr}(t)$ as the processing delay, which is the maximum latency of a node to place its instance since these instances are processed in parallel. Following the M/GI/1 queueing mode, we have

$$L_n^{pr}(t) = \max \left\{ \frac{1}{\mu_v - \sum_{n \in \mathcal{N}} \frac{x_{nv}\lambda_n(t)}{\phi_n}} \right\}_{v \in \mathcal{V}}, \forall n \in \mathcal{N}, \quad (6)$$

where $\sum_{n \in \mathcal{N}} \frac{x_{nv}\lambda_n(t)}{\phi_n}$ is the aggregation traffic of all the instances placed in node $n$. Similarly, the transmission delay will increase as many virtual links (i.e., connections between instances) are embedded on the physical link $(uv)$. Given the bandwidth $L_{uv}$ of link $(uv)$, the transmission delay is calculated as follows:

$$L_{uv}^{tr}(t) = \frac{1}{L_{uv} - \sum_{n \in \mathcal{N}} r_{uv}^n(t)}, \forall u, v \in \mathcal{V}, \quad (7)$$

where $\sum\limits_{n \in \mathcal{N}} r_{uv}^n(t)$ is the aggregated routing rate on link $(uv)$. Hence a placement scheme should guarantee a propagation latency constraint from the source to the destination of the service chain as follows:

$$L(t) = \sum_{n \in \mathcal{N}} L_n^{pr}(t) + \sum_{u,v \in \mathcal{V}} L_{uv}^{tr}(t) \leq \bar{L}, \quad (8)$$

where $\bar{L}$ is the given latency threshold of the service.

*3) Objective function:* In this work, we aim to find a dynamic solution that can place VNFs in the substrate network to minimize the total cost of provisioning resources. Concretely, an optimal solution should minimize the number of instances that can satisfy all requirements of the service chain. Furthermore, it is necessary to optimize the number of instances that is changed in each node during the considered period since placing tasks always consume significant operational cost in terms of resources, configuration and failure [15]. Hence, to minimize the implementation cost, we formulate the objective function as follows:

$$C(t) = \sum_{n \in \mathcal{N}, v \in \mathcal{V}} (\beta_n x_{nv}(t) + \gamma_n [x_{nv}(t) - x_{nv}(t-1)]^+)$$

The first term of $C(t)$ presents the cost of placement instances of the $n$ VNF with the instance cost $\beta_n$. The second term is to calculate the cost of change (i.e., adding/removing some new instances) at node $v$ with the monetary parameter $\gamma_n$.

*4) Joint dynamic routing and placement problem for IoT service chain:* With constraints and objective functions above, we formulate the optimization problem called Joint dynamic placement and routing problem for IoT service chain (JORP).

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{r}} \quad & \sum_{t=1}^{T} \sum_{n \in \mathcal{N}, v \in \mathcal{V}} (\beta_n x_{nv}(t) + \gamma_n [x_{nv}(t) - x_{nv}(t-1)]^+) \\
\text{s.t.} \quad & (1), (2), (3), (4), (5), (8), \\
& (x_{nv}(t) - 1) \sum\nolimits_{v \in \mathcal{V}} r_{vv'}^n(t) \geq 0, \quad (9) \\
& x_{nv}(t) \in \mathbb{N}, t = 1, ..., T, \forall n \in \mathcal{N}, v \in \mathcal{V}, \quad (10) \\
& r_{uv}^n(t) \geq 0, t = 1, ..., T, \forall n \in \mathcal{N}, v \in \mathcal{V}. \quad (11)
\end{aligned}$$

We add another constraint to relate two decision variables in our model by (9). It is to ensure that if the $n^{\text{th}}$ VNF is not located in node $u$, then there is no traffic rate allocated (i.e., if $x_{nv} = 0$ then $\sum_{v \in \mathcal{V}} r_{uv}^n = 0$).

*5) Discussion:* JORP is the MINLP, which is NP-hard in general. The integer allocation variables and the continuous routing rate variables make the problem more challenges to find an optimal solution. In this work, to simplify JORP, we relax it by taking to account the solution of current timeslot $t$ with given the placement scheme of the previous timeslot $t-1$. It means that by observing the current demands and previous placement result, we make the allocation decision at timeslot $t$ to minimize the system cost. Hence, the relaxed problem JORP is rewritten as follows:

$$\begin{aligned}
\text{JORP}' : \min_{\boldsymbol{x},\boldsymbol{r}} \quad & \sum\nolimits_{n \in \mathcal{N}, v \in \mathcal{V}} (\beta_n x_{nv} + \gamma_n [x_{nv} - x'_{nv}]^+) \\
\text{s.t.} \quad & (1), (2), (3), (4), (5), (8), \\
& x_{nv} \in \mathbb{N}, \forall n \in \mathcal{N}, v \in \mathcal{V}, \quad (12) \\
& r_{uv}^n \geq 0, \forall n \in \mathcal{N}, v \in \mathcal{V} \quad (13)
\end{aligned}$$

where $\boldsymbol{x}' = \{x'_{nv}\}_{n \in \mathcal{N}, v \in \mathcal{V}}$ is the given allocation in previous timeslot and time index are removed from the constraints (1), (2), (3), (4) and (8).

Among many optimization approaches to find the answer for an MINLP problem, the BnB algorithm is one of the well-known algorithms in the state of the art. We next present how BnB can find the optimal solution of JORP and discuss the disadvantage of this method in the context of IoT networks.

### C. BnB-based approach for VNF placement problem

We summarize JORP' by

$$\begin{aligned}
& \min && f(\boldsymbol{x}, \boldsymbol{r}) \quad (14) \\
& \text{subject to} && \mathcal{Z}(\boldsymbol{x}, \boldsymbol{r}) \leq 0,
\end{aligned}$$

where every elements of $\boldsymbol{x} = \{x_j\}_{j=1,2,...,|\mathcal{N}| \times |\mathcal{V}|}$ is integer, elements of $\boldsymbol{r}$ are real, and $\mathcal{Z}(.)$ are constraints. Note that, we transform variables from multiple dimension variables into one dimension for ease of perform BnB.

Using BnB method to solve JORP', we find the integer variable under the iterative search-tree method [12]. Intuitively, the root problem in the tree is constructed by relaxing the integer value of JORP'. Then, we iteratively find the integer value of each element of this variable by branching the problem depending on the real value found in the parent node. The child node is created by adding a new constraint to branch the range of the selected element. In this way, the solution of each node $m$ is the upper bound of JORP'.

Mathematically, we present BnB as follows. Let denote $m$ as the pruning node at the $k^{\text{th}}$ iteration and $\boldsymbol{P}_m$ as the relaxed problem of node $m$. In $\boldsymbol{P}_m$, all the integer constraints are relaxed; for example, $\mathcal{C}_m = \{(\boldsymbol{x}, \boldsymbol{r}) : x_j \in \mathbb{N}, j = 1, 2, ..., J\}$ with $J = |\mathcal{N}| \times |\mathcal{V}|$, can be relaxed into $\mathcal{R}_m = \{(\boldsymbol{x}, \boldsymbol{r}) : x_j \in \mathbb{R}, j = 1, 2, ..., J\}$. Hence, the original problem can be relaxed as $\min\{f(\boldsymbol{x}, \boldsymbol{r}) : (\boldsymbol{x}, \boldsymbol{r} \in \mathcal{R}_m)\}$. From the root node $m = 0$ corresponding to the relaxed problem from the original one (14), at each iteration, the relaxed problem is solved to obtain the optimal value $f^*$ and the optimal solution $\boldsymbol{x}^*$ and $\boldsymbol{r}^*$. Since all the integer variables are relaxed, (14) is more tractable. The pruning policy $\pi_m$ is then applied to divide into two following branches. The left child node includes the problem of the parent node and the new constraint as follows:

$$\mathcal{R}_m^l = \mathcal{R}_m \cap \{\boldsymbol{x}, \boldsymbol{r} : x_j \leq x_j^*\}. \tag{15}$$

Thus, the left child problem is formed as follows

$$\boldsymbol{P}_m^l : \min_{\boldsymbol{x},\boldsymbol{r}} \{f(\boldsymbol{x}, \boldsymbol{r}) : (\boldsymbol{x}, \boldsymbol{r}) \in \mathcal{R}_m^l\}. \tag{16}$$

Similarly, the right child node is added by including the parent problem and the new constraint as follows:

$$\mathcal{R}_m^r = \mathcal{R}_m \cap \{\boldsymbol{x}, \boldsymbol{r} : x_j \geq x_j^*\}, \tag{17}$$

$$\boldsymbol{P}_m^r : \min_{\boldsymbol{x},\boldsymbol{r}} \{f(\boldsymbol{x}, \boldsymbol{r}) : (\boldsymbol{x}, \boldsymbol{r}) \in \mathcal{R}_m^r\}. \tag{18}$$

These steps are repeated until every element of $\boldsymbol{x}$ is integer. A node is pruned if its optimal solution $f_m^*$ is less than $f_{min}$ (*bounding condition*), or $\boldsymbol{P}_m$ is infeasible (*infeasible condition*), or finally, all the elements of $\boldsymbol{x}$ are integer (*integral condition*).

In order to reduce the complexity of BnB, in the next part, we present a learning model based on DNN and the imitation theory that can imitate the procedure of BnB to efficiently find the near-optimal solution of JORP.

## III. L-JORP: Learning for VNF placement in IoT service chain implementation

Consider the search tree of BnB, the final solution belongs to a specific branch in the tree. Thus, if we can recognize such a good branch and prune all others, the search space can be reduced significantly. We design the architecture of DNN that can represent specific characteristic of JORP in order to learn pruning rules for searching the optimal solution based on BnB algorithm. Furthermore, to "steer" the searching into potential branches, we apply the imitation theory, DAgger [13] in the training phase.

### A. DNN: A supervisor learning model for classification

In this model, we design the input as a feature vector of the nodes and the output as a binary value {0,1} corresponding to *prune* and *preserve*. We employ a $K-$layer Perceptron neural network [16]. At the first step, we encode the optimization problem by representing the features of binary variables and constraints and the objective value as shown in Fig. 2. The relationship between the binary variables and the constraints is implied by the link $e_{i,j}$, i.e., if the $v[i]$ belongs to the constraint $c_j$, then $e_{i,j} = 1$, otherwise $e_{i,j} = 0$. The value of each node in the constraint layer is activated (i.e., it equals to 1) if the input values satisfy the constraint, otherwise it is deactivated.

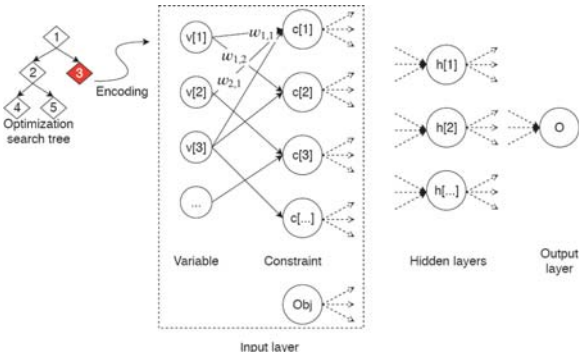



Fig. 2: Deep neural network for learning to branch.

The features of the input layer will propagate to the next layer by the Rectified Linear Unit function $\mathbf{ReLU}(.)$, (e.g., we use the function $\max(0, .)$). Thus, the output of the $k^{\text{th}}$ layer is as follows $\boldsymbol{g}^k = \mathbf{ReLU}(\boldsymbol{w}^k \boldsymbol{g}^{k-1} + \boldsymbol{b}^k)$, where $\boldsymbol{w}^k$ and $\boldsymbol{b}^k$ are the learning and bias parameters of the $k^{\text{th}}$ layer, respectively.

Finally, the output is calculated as the probability which is normalized as follows:

$$O[i] = \frac{\exp(g_i^K)}{\sum_{j=0,1} \exp(g_i^K)}, i = 0, 1. \tag{19}$$

We design the output layer with two nodes to classify input nodes that belongs to prune class or preserve class.

Furthermore, we use the weighted cross entropy loss function as follows $\mathcal{L} = -\sum_{i=1,2} w_i y_i \log(O_i)$, where $\boldsymbol{y}$ is the label vector to measure the loss compared to the output $\boldsymbol{O}$.

### B. Training phase

The proposed neural network can learn policies in the optimization search tree via a supervised training phase. However, how to train the neural network even for the medium setting of VNF placement problem in IoT networks should be carefully considered because of the huge number of nodes in the optimization search tree. For the supervised learning process, a pair of input and output values is used to train in each iteration. It means that each optimization of each node has to be solved to apply the pruning rules.

We inspire from the collecting data, DAgger algorithm [17], to propose an algorithm to collect data automatically by using an optimization solver for training. We run the training phase with the maximum $U$ iterations. At the beginning, the policy $\pi^0$ is added to $\Pi$ by setting the pruning policy if the root node is infeasible and the preserve policy if it has a solution. For each trajectory node in the tree, we let the system prune the node according to learning policy $\Pi$ (Lines 7-9). Then, the best policy of pruning $\pi^*(m)$ is given to "steer" the direction of branching, but the system is ignoring them and simply add this expert result to the data set for training (Lines 10-12). From now, we will train a new policy to make the system remember the correctness $\pi^*(m)$ (Line 13). The branching step (Lines 14-22) is similar to the BnB method in adding children nodes into the stack for the next iteration. Depending on the problem, $U$ might be adjusted during the training of the neural network until it approaches to the good performance.

**Algorithm 1:** Imitation learning based algorithm for solving JORP.

Initialization:
$\mathcal{S} := \{\}, f^* := \infty, \Pi := \{\pi_0\}, u = 0, m = 0, \mathcal{D} \leftarrow \emptyset$;
**while** $u \leq U$ **do**
  u:=u+1;
  Set the relaxed problem JORP to node $m$;
  $\mathcal{S}.push(m)$;
  **while** $(m \leftarrow S.pop())!=$ *Null* **do**
    Solve the relaxed problem of $m$ to obtain $(\boldsymbol{x}_m^*, \boldsymbol{r}_m^*)$ and $f_m^*$;
    Classify node $m$ by DNN with the trained policy $\Pi$: $\Phi(m)$;
    Get dataset $\mathcal{D}_u = \{(m, \pi^*(m))\}$ of visited nodes by a decision $\pi_u^*$ given by an expert;
    Add a new policy to $\Pi = \Pi \cup \pi^*(m)$;
    Aggregate datasets $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_u$;
    Train classifier on $\mathcal{D}$;
    **if** $\Phi(m)$ *is preserve* **then**
      **if** $f^* > f_m^*$ **then**
        Branching to $m^l$ and $m^r$ corresponding to real element $x_j$;
        $\mathcal{S}.push(m^l, m^r)$;
      **end**
    **end**
    **if** $\{x_j^* \in \mathbb{N} | \forall j\}$ **then**
      $f^* := \min(f^*, f_i^*)$;
    **end**
    **else if** $u$=$U$ **then**
      *#Increase threshold if* $x_j \notin \mathbb{N}$
      U++;
    **end**
  **end**
**end**

### C. Testing phase

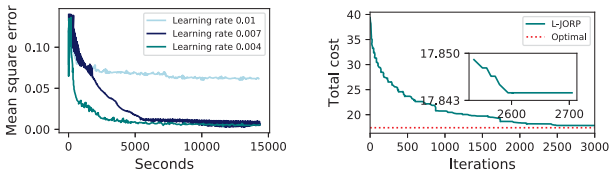
In the testing phase, we use the pruning policies learned from the training phase to scan the optimization search tree instead of using the regular policy of the BnB algorithm. If the selected is the leaf node, a convex solver is invoked to solve since the sub-problem now becomes convex with given relaxed variables. Thus, with the set $\Pi$ of learning policies, the neural network tries to find the optimal solution with minimum number of nodes expanded and reducing the computation. To do that, an optimal node is kept to compare with the solution in each iteration.

The execution of this algorithm is similar to BnB; however, instead of using the pruning rules, we branch and prune nodes by the neural network, which is denoted by $\Phi(m)$.

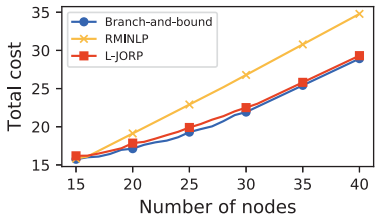
## IV. Numerical results

### A. Settings

*1) Optimization settings:* We consider a general implementation with a hierarchical smart city architecture including 3 cloud nodes and 20 distributed edge nodes. Moreover, to verify the performance of the proposed algorithm, a varying setting




(a) Learning rate evaluation. (b) Optimal value trajectory.




(c) Optimalilty gap comparison.

Fig. 3: Convergence evaluation.

of nodes and VNFs in a service chain is also considered in our work. For each service chain, the source and destination nodes of the IoT service chain are given. In particular, IoT requests from sensors will be aggregated by a gateway as a source node, and at the end of the service chain, the dispatching function connected to end devices is the destination node.

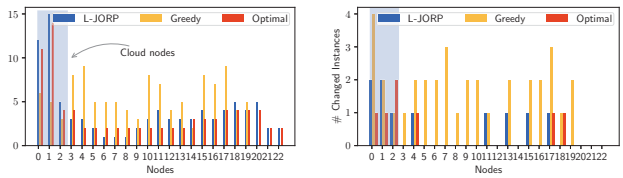
In order to evaluate the benefit of the edge-cloud based model, we set the upper bound latency of the service in range from 30 ms to 150 ms. Meanwhile, the propagation latency between nodes is in the range between 30 to 100 ms, which also implies the distance between them.

*2) Learning settings:* Our learning framework is developed by Python with supporting from Pytorch to create the neural network. For the neural network, we design three hidden layers with the setting as 32x64x32. We set $6.0 \times 10^-3$ as the threshold of mean square error for training.

### B. Results

*1) Convergence:* We first present the convergence result of the neural network by varying learning rate. Mean square error rate is used to analyze the convergence performance of the accuracy of the neural network as shown in Fig. 3b. At the learning rate 0.01, the neural network can reach the convergence very fast; however, its mean error rate is the highest compared to other settings. We observe that with the learning rate 0.004, our designed neural network can obtain a good convergence after 5000 iterations while the mean square error fluctuate between $5.5 \times 10^{-3}$ and $6.1 \times 10^{-3}$. For further simulation results, we fix this learning rate setting to execute the learning framework in both training and testing phases.

We next evaluate the performance of L-JORP as shown in Fig. 3b. We simulate the performance under the setting with 20 nodes and 30 VNF instances. The comparison is taken into account the optimal solution solving by BnB algorithm and L-JORP. Using BnB (considered as the optimal solution), this algorithm needs to go through 18347 nodes to find the optimal solution in 352.57 seconds. Meanwhile, L-JORP can reach the




(a) Number of used instances. (b) Number of changed instances.

Fig. 4: Evaluation of JORP over long run (5 timeslots).

optimal node with only 2931 nodes as shown in Fig. 3b. The trajectory of the optimal cost in each step is also zoomed in to illustrate the behavior of the cost when it reaches to the convergence. Especially, L-JORP spends only 15.24 seconds to obtain the optimal solution since we use the lookup table to get the solution directly whenever the optimization nodes are already saved in our database.

Finally, we conduct the optimality gap between our framework and others by scaling up the network setting from 15 to 40 nodes. For each setting, we have to do the training before executing the evaluation. In this evaluation, we compare L-JORP with the relaxed mixed-integer nonlinear programming (RMINLP) based algorithm [18] and BnB algorithm in solving JORP. Fig. 3c illustrates the significant gap between RMINLP and BnB when scaling up the system. Meanwhile, L-JORP follows strictly the trend of the optimal line from small to large settings. Thus, the result demonstrates a certain capability of L-JORP in finding the optimal solution for JORP under different network settings.

*2) The efficiency of* JORP*:* We conduct the following experiment to evaluate the JORP's efficiency. As shown in the objective function of JORP, our model is to minimize the implementation cost in terms of number of instances that need to be placed and the changes of allocation. During 5 timeslots, Greedy approach attempts to place VNF instances to satisfy service constraints. Since it does not consider the optimal placement, the number of used instances is highest compared to others with 112 instances. Especially in Fig. 4a, it only places a few instances into cloud nodes with 14 VNF instances while Optimal can place 29 instances. JORP can reduce 19.6% the number of instances during 5 timeslots compared to Greedy baseline. Furthermore, the distribution of instances in the system of JORP shows the load-balancing ability when it can mitigate the unbalanced phenomenon as in case of Greedy (i.e., many instances are placed in a node while other nodes are low-utilized).

In addition, the efficiency of JORP is demonstrated in Fig. 4b by the number of changed instances. Concretely, during 5 timelots, Greedy gets the highest of change compared to JORP and Optimal with 35 instances while JORP can reduce up to 65.4% the result of Greedy during the considered time. The main reason of the highest change in the Greedy is from its unawareness of previous placement scheme.

## V. Conclusion

In this article, we consider the joint dynamic routing and VNF placement problem for IoT service implementation. We formulate JORP to minimize the operational cost of implementing IoT services that is subject to all the resource and latency constraints. The problem is formulated regarding the long run of the system with the time-varying workload. In order to find the solution for JORP, which is NP-hard, we proposed the learning model designed by a DNN to solve JORP under the procedure of BnB method. We further proposed a training model using the imitation learning to aggregate automatically data during the training phase. We show that JORP outperforms baselines with outstanding processing time.

## References

[1] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[2] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, "Enorm: A framework for edge node resource management," *IEEE Transactions on Services Computing*, 2017.

[3] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," *arXiv preprint arXiv:1609.01967*, 2016.

[4] Docker. [Online]. Available: https://www.docker.com/

[5] OpenStack Neat: A Framework for dynamic consolidation of virtual machines in OpenStack Clouds a blueprint. [Online]. Available: http://www.cloudbus.org/reports/OpenStack

[6] Kubernetes. [Online]. Available: https://containership.io/

[7] Cloud iot edge - google cloud. [Online]. Available: {https://cloud.google.com/iot-edge}

[8] A. Leivadeas, G. Kesidis, M. Ibnkahla, and I. Lambadaris, "Vnf placement optimization at the edge and cloud," *Future Internet*, vol. 11, no. 3, p. 69, 2019.

[9] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier iot fog networks: A joint optimization approach combining stackelberg game and matching," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1204–1215, Oct 2017.

[10] J. Wang, H. Qi, K. Li, and X. Zhou, "Prsfc-iot: a performance and resource aware orchestration system of service function chaining for internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1400–1410, 2018.

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

[12] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer programming*. Springer, vol. 271.

[13] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[14] J. F. Riera, E. Escalona, J. Batall, E. Grasa, and J. A. Garca-Espn, "Virtual network function scheduling: Concept and challenges," in *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*, June 2014, pp. 1–5.

[15] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

[16] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[17] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[18] Y. Cheng, M. Pesavento, and A. Philipp, "Joint network optimization and downlink beamforming for comp transmissions using mixed integer conic programming," *IEEE Transactions on Signal Processing*, vol. 61, no. 16, pp. 3972–3987, 2013.