Reinforcement Learning Based Approach for Virtualized Face Detection at the Edge

Makhlouf Hadji, Selma Khebbache, Mohammed Khalidi Idrissi

Reinforcement Learning Based Approach for Virtualized Face Detection at the Edge

Selma Khebbache*, Makhlouf Hadji*, Mohamed-Idriss Khaledi *
*Institut de Recherche Technologique SystemX, Saclay, France
Emails: {selma.khebbache, makhlouf.hadji, mohamed-idriss.khaledi}@irt-systemx.fr

***Abstract*—Real-time requirements in video streaming and processing are increasing and represent one of the major issues in industry 4.0 domains. In particular, Face Detection (FD) use-case has attracted the interest of industrial and academia researchers for various applications such as cyber-physical security, fault detection, predictive maintenance, etc. To ensure applications with real time performance, Edge Computing is a good approach which consists in bringing resources and intelligence closer to connected devices and hence, it can be used to cope with strong latency and throughput expectations. In this paper, we consider optimal routing, placement and scaling of virtualized face detection services at the edge. We propose an edge networking approach based on Integer Linear formulation to cope with small problem instances. A reinforcement learning solution is proposed to address larger problem sizes and scalability issues. We assess the performance of our proposed approaches through simulations and show advantages of the reinforcement learning approach to converge towards near-optimal solutions in negligible time.**

***Index Terms*—Face Detection, Optimization, Q-Learning, Edge Networking**

I. Introduction

Industry 4.0 sites' deployment is growing and necessitates new needs and strong expectations in terms of performance, ultra low latency, security, etc. In addition, with a massive deployment of Industrial IoTs (IIoTs) in Industry 4.0, a particular issue on video streaming and processing has attracted the interest of industrial and academia researchers to cope with intrusion detection in cyber physical systems (CPS), fault detection in large industrial systems, predictive maintenance in complex production systems, etc. Indeed, industrial players require cost-efficient deployment solutions for interconnected cameras and video analysis systems to reduce their total (CAPEX and OPEX) costs, minimize human interventions using zero-touch solutions. Hence, and to reach these objectives, virtualized techniques are being privileged for this purpose, where both the networking functions are virtualized, using Virtualized Networks Functions (VNF), and application functions are flexibly developed for being deployed on edge nodes.

To attend the previous cited objectives and performance in industry 4.0, we need new orchestration solutions to distribute computing resources, storage, network connectivity, data analytics through interconnected nodes called Edge clouds. Indeed, edge computing and networking can be used to bring the necessary resources closer to connected devices and IoTs. This may strongly reduce the latency when improving the analytics performance through intelligent and distributed computing and algorithms.

978-1-6654-4005-9/21/$31.00 ©2021 IEEE European Union

In industry 4.0, there exist relevant and challenging use-cases which attracted the intention and interests of industrial players looking for augmented and intelligent cyber-security solutions, adaptive and rapid approaches for fault detection or predictive maintenance, etc. In this work, we focus on a particular use-case based on Face Detection represented by virtualized functions that will be detailed in the sequel. In fact, deploying a virtualized face detection solution in a critical industrial site which necessitates accuracy of the analytics, and real time decision making, is relevant for the safety and security of this site.

We consider virtualized face detection use-case which is composed by virtualized functions in a given service chain. These virtualized functions should be deployed in a distributed manner to optimize the limited network resources and compute resources on the edge nodes. These nodes are equipped with limited processing capabilities, and are interconnected with different communications techniques such as Ethernet, Bluetooth, Wifi, Lora, etc. This heterogeneity of the processing capabilities and communications solutions makes the orchestration and management of the deployed network services harder. We need to investigate efficient optimization of these resources in near real time to attend face detection solutions with good performance (convergence time, reduced CAPEX and OPEX cost, etc.).

Our contribution in this paper consists to address deeply the aforementioned research challenges by proposing mathematical modeling and formulations. These approaches can be summarized as follows:

- An exact approach based on integer linear programming : this method is based on the description of the convex hull of the addressed virtualized face detection optimization problem in edge infrastructures.
- To cope with scalability issues, we propose an approximation approach based on Reinforcement Learning.

Note that the novelty of our approaches will be clearly addressed in next sections and their efficiency will be highlighted and compared to the state of the art addressing close research challenges.

The remainder of this paper is organized as follows: next section (Section II) addresses the related work on face detection research challenges, edge computing and networking and

optimization. Section III describes the details of the considered optimization problem, provides hints on the problem's complexity and formalizes two new mathematical approaches based on ILP and RL. Our approaches will be evaluated using simulations in Section IV. We conclude the paper in Section V.

II. Related work

Reference [1] is one of the closest work to our paper. it addresses a joint optimization of power consumption of the CPU and the Wireless Network Interface Card (WNIC) of mobile devices while streaming high quality videos. We recall that the two major energy consuming components in mobile video streaming services are the CPU (that supports video data decoding) and the network interface (that supports data communication). Hence, authors of this paper proposed a joint optimization scheme for improved energy efficiency supporting mobile video streaming services . This is based on the adjustment of the number of video chunks to be downloaded and decoded in each packet. However, they only considered joint optimization of CPU and energy consumption for one mobile device. In our work, we consider a joint optimization of CPU and bandwidth in a IoT network with multiple physical objects and where the number of paths between each couple of nodes can be exponential.

Authors of paper [2] discussed the problem of real time video analytics in an edge computing environment. Recall that real-time video analysis is used nowadays in several domains (traffic control, surveillance and security, retail store monitoring..) and because of the high data volumes, compute demands and latency requirements, cameras represent the most challenging of "things" in IoT, and large-scale video analytics may well represent the killer application for edge computing. The authors proposed a real-time video analytics system with low resource cost to produce outputs with high accuracy. The proposed approach can be used in several application such as self-driving and smart cars, etc. Nevertheless, the proposed approach do not consider a joint optimization for different resources such as CPU and bandwidth when satisfying latency expectations.

Paper [3] proposes a mathematical optimization approach over virtual network services for better scaling, placement and routing on a physical substrate. Network services (video streaming, online gaming) are placed and deployed in the network based on fixed predefined descriptors. With the large number of degrees of freedom for finding the best adaptation, deciding scaling, placement, and routing can result in sub-optimal decisions for the network and for the running services. They proposed JASPER (Joint optimization of scaling, placement and routing) in which, each network service is described by a service template containing information on the component network services, their interconnection, and resources requirements. Hence, the solutions of reference [3] are applied using IETF use-cases and examples for an embedding of virtual networks on physical substrates. In our work, we consider limited IoT nodes in terms of available resources, which makes the problem harder and finding feasible solutions more complicated.

Another reference dealing with CPU and bandwidth optimization is given by [4]. Authors of this paper proposed a technique to manage computation offloading in a local IoT network under bandwidth constraints. They proposed approaches to separately optimizing CPU and Bandwidth resources, which can lead to sub-optimal solutions of the problem. We propose a joint optimization of the two mentioned limited resources and discuss the scalability of our approaches for large instances.

In the context of VNF orchestration, the Stratos project [5] proposed a detailed architecture to orchestrate VNFs outsourced to a remote cloud taking considering traffic engineering, VNFs scaling, etc. Nevertheless, this project has addressed only VNF placement solution based only on workloads, and then, chaining constraints are not considered.

References [6] and [7] propose mathematical models for the VNF chains placement with routing constraints. The proposed models, however, describe a limited number of linear constraints and can only characterize a small portion of the problem convex hull. The proposed exact solutions do not scale for large problem instances. We need deeper modeling that can characterize better (completely) the convex hull of the VNF placement and chaining problem to find near optimal solutions in few seconds and scale with problem size. We propose a competitive graph theory approach with the desired properties of converging quickly to near optimal solutions even for large problem instances.

In our paper, we propose to investigate machine learning techniques based on Reinforcement Learning to reach near optimal solutions of the virtualized face detection optimization problem in edge infrastructures. Our objective is to train RL models to improve the quality of the solutions compared to the results of the first paper addressing this problem (see reference [8]).

III. Mathematical formulation

Before going through the mathematical details of the problem description, we propose to describe the system model and set up the different parameters of our approaches.

A. System model description

We consider an industrial site or a restricted area with various cameras collecting data for monitoring and analytics purposes. These cameras are differently interconnected using wifi, ethernet, etc. and hence are forming a heterogeneous network with limited amounts of bandwidth and processing capabilities (represented by CPU cores in our work). The virtualized face detection service is composed of a chain of 4 virtual functions (denoted by $f_1 \rightarrow f_2 \rightarrow f_2 \rightarrow f_4$) equivalent to Service Function Chains as described by [9]. In the virtualized face detection use-case, the four mentioned virtual functions are given as follows:

1) f_1 (**Face Detection**): detects and extracts in frames all faces in the restricted area.

2) f_2 (**Face compression**): It characterizes each face by limited number of features
3) f_3 (**Face recognition**): compare the extracted features to a given database
4) f_4 (**Database enhancement**): for an Face recognition with a high reliability, the new corresponding features are added to the database to improve the quality of future identification

Figure 1 represents our system model for a virtualized face detection use-case represented by a chain (green graph in Figure 1). The physical graph (blue graph) represents a network of interconnected cameras to capture videos and images and may be equipped by a processing node. Cameras are interconnected and the edge network also incorporates processing servers.

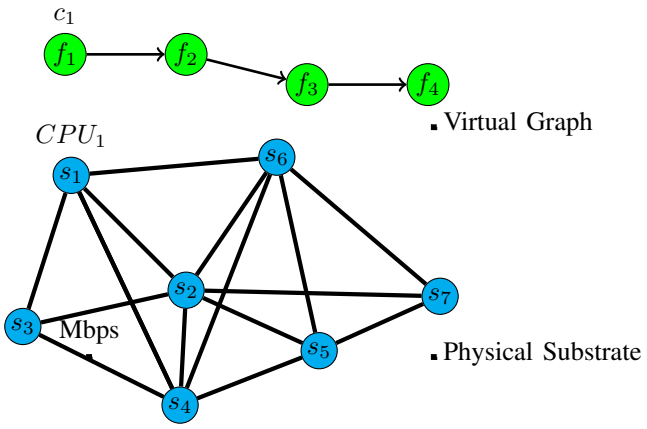

Fig. 1: Virtualized Face Detection system model: a graph-based representation

B. Problem description and modeling

Using the graph representation of Figure 1, the virtualized face detection problem in an edge network with restricted compute and networking resources, is equivalent to find an optimal mapping of the virtual graph (see green graph of Figure 1) on the physical graph (blue graph of Figure 1) when optimizing the aforementioned resources, to reach higher performances.

Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be the physical graph representing the edge nodes substrate (interconnected cameras/sensors and processing nodes), where $|\mathscr{V}|$ and $|\mathscr{E}|$ represent the network sizes in terms of number of nodes and edges, respectively. Let $\mathscr{G}_v = (\mathscr{V}_v, \mathscr{E}_v)$ be the virtual request graph composed by 4 chained VNFs.

Each physical edge node j_1 has a limited CPU processing noted by C_{j_1}. Each physical edge (or link) (j_1, j_2) has a limited amount of bandwidth noted by B_{j_1,j_2}. If there is no edge between j_1 and j_2, then the amount of available bandwidth between these two nodes is represented by the smallest available bandwidth on the path between j_1 and j_2. Each virtual arc (i, i') has a request of $b_{ii'}$ of necessary bandwidth.

We investigate new algorithms to reach optimal trade-offs between the two criteria of processing and bandwidth with respect to latency requirements.

C. Problem complexity

The addressed problem of virtualized face detection optimization and mapping is NP-Complete. More details on the proof can be found in [8].

D. Integer Linear Programming (Exact) approach

We consider $min_{j_1,j_2,k}$ as the minimum bandwidth available in the k^{th} path between physical nodes j_1 and j_2. Moreover, for the sake of clarity, we introduce $\mathscr{P}(j_1, i)$ as the set of all physical nodes j_2 such that there exists a shortest path between nodes j_1 and j_2 in which the minimum available bandwidth (on its arcs) is at least equal to the required amount of bandwidth (in the virtual graph). We define $b_{i,i+1}$ as the bandwidth needed on the arc $(i, i+1)$ of the virtual graph.

We propose a short description of the decision variables used in our optimization:

- x_{i,j_1} is a binary variable, equals to 1 if the virtual node/function f_i is deployed on a physical node j_1, and 0 otherwise.
- $y_{(i,i+1);(j_1,j_2,k)}$ is a binary variable equals to 1 if a virtual arc $(i, i+1)$ is hosted on a physical path k joining two physical nodes j_1 and j_2, and 0 otherwise.
- H_{j_1} is a binary variable equals to 1 if the physical node j_1 hosts at least one virtual function, and 0 otherwise.

The objective function of our optimization consists in finding a trade-off between the total CPU consumption and the allocated bandwidth resources:

$$\max \quad \mathscr{F} = \sum_{j_1 \in \mathscr{V}} \left(C_{j_1} H_{j_1} - \sum_{i \in \mathscr{V}_v} c_i x_{i,j_1} \right) + \tag{1}$$

$$\sum_{i \in V_v} \sum_{j_1 \in \mathscr{V}} \sum_{j_2 \in \mathscr{P}} \sum_{k \in \mathscr{K}} \left(\min_{ij_1j_2k} -b_{(i,i+1)} \right) y_{(i,i+1)(j_1,j_2,k)}$$

We propose a constrained optimization to reach the optimal value of (1):

$$\sum_{j_1 \in \mathscr{V}} x_{i,j_1} = 1, \forall i \in \mathscr{V} \tag{2}$$

Constraint (2) guarantees that each virtual node, i.e., a function f, is deployed on exactly one physical or edge node.

$$\sum_{i \in V_v} c_i x_{i,j_1} \leqslant C_{j_1} H_{j_1}, \forall j_1 \in \mathscr{V} \tag{3}$$

Constraints (3) guarantee the non violation of available CPU resources at edge nodes.

$$\sum_{i \in V_v} x_{i,j_1} \geqslant H_{j_1}, \forall j_1 \in \mathscr{V} \tag{4}$$

Constraints (4) minimizes the total number of edge nodes to be solicited in the final solution.

$$x_{i,j_1} \leqslant \sum_{j_2 \in \mathscr{P}(j_1,i)} x_{i+1,j_2}, \forall i \in \{1,2,3\}, \forall j_1 \in \mathscr{V} \quad (5)$$

Constraints (5) are guaranteeing the necessary chaining of the virtual graph.

$$\sum_{j_1 \in \mathscr{P}(,)} \sum_{k \in \mathscr{K}(j_1,j_2,i)} y_{(i,i+1)(j_1,j_2,k)} = x_{i+1,j_2}, \forall i \in \{1,2,3\}, \forall j_2 \in \mathscr{V} \quad (6)$$

$$\sum_{j_2 \in \mathscr{P}(j_1,i)} \sum_{k \in \mathscr{K}(j_1,j_2,i)} y_{(i,i+1)(j_1,j_2,k)} = x_{i,j_1}, \forall i \in \{1,2,3\}, \forall j_1 \in \mathscr{V} \quad (7)$$

Constraints (6) and (7) are used to guarantee that if a virtual node i is deployed on a physical node j_1, i.e. $x_{i,j_1} = 1$, and the virtual node $i+1$ is hosted by a physical node j_2, i.e. $x_{i+1,j_2} = 1$, then the virtual arc $(i, i+1)$ should be deployed on the k^{th} physical path starting from node j_1 to node j_2, i.e. $y_{(i,i+1);(j_1,j_2,k)} = 1$.

$$\sum_{j_1 \in \mathscr{V}} \sum_{j_2 \in \mathscr{P}(j_1,i)} \sum_{k \in \mathscr{K}(j_1,j_2,i)} y_{(i,i+1)(j_1,j_2,k)} = 1, \forall i \in \{1,2,3\} \quad (8)$$

Constraints (8) guarantee that each virtual arc $(i, i+1)$ is deployed on exactly one **physical path**.

$$\sum_{i \in 1,2,3} \left(y_{(i,i+1)(j_1,j_2,k)} + y_{(i,i+1)(j_2,j_1,k)}\right) b_{(i,i+1)} \leqslant min_{j_1 j_2 k}$$
$$\forall l \in \{1,2,3\}, \forall j_1 \in \mathscr{V}, \forall j_2 \in \mathscr{P}(j_1,l), \forall k \in \mathscr{K}(j_1,j_2,l) \quad (9)$$

Constraints (9) impose that the bandwidth of a feasible physical path between nodes j_1 and j_2 must exceed the sum of all hosted $b_{i,i+1}$ on that path.

Table I summarizes optimization parameters and variables used in our mathematical formulation.

TABLE I: Mathematical formulations' parameters

Parameters	***Definition***
x_{ij_1}	binary variables indicating if f_i is hosted in j_1
$y_{i,i';j_1,j_2}$	a binary variable indicating if the virtual arc (i,i') is hosted on the physical path between j_1, j_2
H_{j_1}	a binary variable indicating if node j_1 is solicited
c_i	the requested CPU amount by VNF_i
C_{j_1}	The available CPU amount in node j_1
$K_{(j_1,j_2,i)}$	a set of all available and feasible (a path with the minimum bandwidth on all its edges is higher than the requested bandwidth for the arc $(i,i+1)$) paths between nodes j_1 and j_2 for the virtual arc $(i,i+1)$
$min_{i,j_1,j_2,k}$	Minimum available bandwidth on the k^{th} feasible path between physical nodes j_1 and j_2 for the virtual arc $(i,i+1)$

E. Reinforcement Learning based Approach

a) Problem formulation with Markov Decision Process (MDP): In the following we propose a new approach based on Reinforcement Learning to cope with scalability issues and will be benchmarked by the exact approach given by the ILP algorithm. We formulated the problem as a Markov Decision Process (MDP). In an MDP, a decision-maker noted by **agent** is interacting with the environment represented by the described physical infrastructure (blue graph of Figure 1). These interactions occur sequentially over time. At each time step, the agent will obtain some representation of the environment's state. Given this representation, it selects an action to be considered. The environment is then transitioned into a new state, and the agent receives a **reward** according to its previous action.

The process of selecting an action from a given state, transitioning to a new state, and receiving a reward is occurring iteratively and creates a trajectory showing the sequence of states, actions, and rewards. Throughout this process, the agent maximizes the total amount of rewards it receives. In other words, the agent maximizes the cumulative rewards it receives over time. Figure 2 illustrates this mechanism.

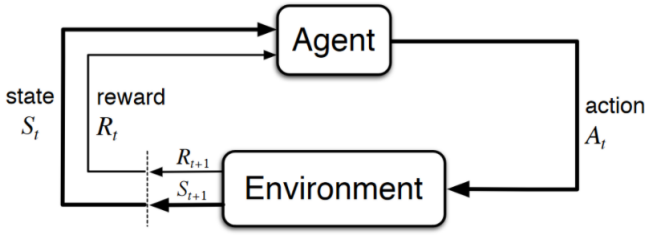

Fig. 2: The agent-environment interaction in an MDP (source: [10])

To mathematically model the above mentioned description, we consider a set of states S, a set of actions A, and a set of rewards R. These sets have a finite number of elements described as follows:

1) **States**: a state $S_t \in S$ represents a physical node in our infrastructure and precisely the node where the agent is placed in at time step t. We claim that the agent is in state j at time step t referred as S_t^j, if it is placed on physical node j at time step t.
2) **Actions**: an action $A_t \in A$ represents the next physical node starting from the state S_t.
3) **Rewards**: a reward R_t is obtained at time step t after selecting the next physical node j and it consists of

the remaining CPU value of node j plus the remaining bandwidth value on the selected path.

4) **Transition Probability**: Since the sets S and R are finite, random variables S_t and R_t have well-defined probability distributions. These distributions depend on the preceding state and action that occurred in the previous time step $t-1$. However, these are not easy to be obtained in a real network. Alternatively, RL comes as a solution because it can learn from previous experiences in a trial-and-error fashion, and choose the appropriate actions without explicit state transition probability.

b) Q-Learning algorithm: In the following, we describe our Q-Learning algorithm modified and adapted to our problem. For each episode, we deploy the virtual function f_1 on the physical node that has the maximum CPU capacity. Then, we iteratively go through three time steps. The first one is to deploy f_2 on the physical graph, the second one is to deploy f_3, and the third is to deploy f_4. To describe the process at every episode, and after selecting the corespondent physical node (the best in terms of CPU) that will host f_1, the agent receives for every time step $t = 1, 2, 3$ some representation of the environment's state $S_t \in S$ which represents the physical node where the agent is placed in at time t with updated CPU and bandwidth. Based on this state, the agent selects an action $A_t \in A$ that represents the next node to consider from S_t. This provides the state-action pair (S_t, A_t) and we can define $Q(S_t, A_t)$ as the state-action value function which represents the reward when starting in state S_t and selecting action A_t at time step t.
The decision of choosing an action in a certain state at time step t is characterized by a policy $\pi(S_t) = A_t$. Our goal is to find an optimal policy that maximizes the value of the state-action function.

$$\pi_{opt}(S_t) = {}_{A_t \in \mathcal{A}}\, Q(S_t, A_t), \forall t \in \{1, 2, 3\} \tag{10}$$

c) Description of Q-Learning function: Note that, given a step, a state, and an action, the Q-function provides the expected reward from taking the given action in the given state following the given policy at time step t. At each episode, our Q-Learning algorithm iteratively updates the Q-values of our three-dimensional Q-table until it converges to the optimal one. This Q-table is composed of three tables: the first contains the Q-values that correspond to the action of hosting f_2, the second is for hosting f_3 and the third is for hosting f_4. These Q-values are updated at each time step as follows:

$$\underbrace{\mathrm{Q}(S_t, A_t)}_{\substack{\text{New}\\ \text{Q-Value}}} = \underbrace{Q(S_t, A_t)}_{\substack{\text{Current}\\ \text{Q-Value}}} + \alpha \Big[\underbrace{R(S_t, A_t)}_{\text{Reward}} + \gamma \overbrace{\max_{A_t} Q'(S'_t, A'_t)}^{\substack{\text{Maximum predicted reward, given}\\ \text{new state and all possible actions}}} - Q(S_t, A_t) \Big] \forall t \in \{1, 2, 3\} \tag{11}$$

d) Exploration and Exploitation steps:: In our Q-Learning approach, **Exploration** step is the operation of exploring the environment to find out information about it and **Exploitation** represents the operation of exploiting the information that is already known about the environment in order to maximize the return.

Hence, if the agent is able to explore the environment, it will have the opportunity to find the group of physical nodes that would maximize the remaining CPU and bandwidth capacities in our physical graph. Nevertheless, if it only explores the environment with no exploitation, then it will miss out on making use of known information that could help to maximize the cumulative reward. Therefore, we need a balance of both exploitation and exploration, which should be simultaneously performed. To reach this objective, we use the epsilon greedy strategy defined below.

e) Epsilon-greedy strategy:: To attend the exploitation and exploration trade-off, we use the epsilon greedy strategy. We define an exploration rate ϵ that we initially set to 1. This exploration rate is the probability that the agent will explore the environment rather than exploiting it. With $\epsilon = 1$, it is 100% certain that it will start out by exploring the environment. As the agent learns more about our environment, at the end of each new episode, ϵ will decay by some rate that we set, so that the likelihood of exploration becomes less and less probable as the agent learns more and more about the environment. At the end, this agent will become "greedy" in terms of exploiting the environment once it has the opportunity to explore and learn more about it. The main procedure of our Q-Learning algorithm is represented as follows (see Algorithm 1).

Algorithm 1 Q-Learning Algorithm

```
Q(S_t, A_t) ← 0 ∀S_t ∈ 𝒮, ∀A_t ∈ 𝒜, ∀t ∈ {1,2,3};
Set up Q-Learning hyperparameters;
while episode ⩽ numberOfEpisods do
    Choose the best node in terms of CPU capacity to host f_1 and update the physical graph;
    while t ⩽ 3 do
        if Exploration then
            Select randomly the next node able to host the next virtual function f_{t+1};
        end if Choose A_t = _{A_t} Q(S_t, A_t);
        Perform A_t and get the reward and the new state;
        Update Q(S_t, A_t) according to equation (III-E0c);
        Update CPU and Bandwidth on the physical graph;
    end while
    Do the exploration rate decay according to epsilon-greedy strategy;
end while
```

IV. Numerical Results

Our proposed approaches are evaluated through a Python implementation using CPLEX [11] solver for the exact approach. We use a laptop with 8Gb of RAM and 2.7Ghz of CPU. The following metrics are used to quantify the efficiency of the two proposed approaches (exact and Q-Learning):

1) **Gap** between the objective functions of the proposed approaches: this metric is provided to measure the

efficiency of the RL algorithm benchmarked by the exact method. It is given as follows:

$$Gap(\%) = \frac{ILP_{Sol} - RL_{Sol}}{ILP_{Sol}} \times 100$$

2) **Convergence time**: The necessary time to converge to a (near) optimal solution
3) **Scalability**: The ability of the algorithms to scale and provide a solution within an acceptable convergence time.
4) **Rejection rate**: it represents the capacity of our proposed approaches to process virtual requests in given time window.

To assess the performance of our proposed approaches, we run various scenarios corresponding to the generation of multiple physical graphs and virtual requests. We propose a dynamic simulator addressing requests arrivals according to a Poisson process. For each scenario in our simulations, we represent each point by the average value of 50 runs of different collected metrics of the two proposed approaches. Simulation settings will be detailed in the sequel.

A. Simulation settings

In the following, we address simulations with dynamic setting where demands for virtualized face detection chains arrive according to a Poisson process with inter-arrival times (noted by λ^{-1}) and departures are represented by exponential distribution with a parameter (lifetimes) represented by μ^{-1}, also noted by service rate. This dynamic setting allows to compute the system capacity, in terms of rejection rate. Hence, we introduce $\rho = \frac{\lambda}{\mu}$ to assess these performances. Note that the event of request rejection occurs when a new request arrives while the system still processing a large number of previous requests. This is modeled in our case by a buffer for requests of limited size (10 in the numerical results). When this buffer is full, new arriving requests are rejected. This rejection is thus related to the algorithm's necessary convergence time: The faster the convergence, the lowest is the probability of buffer overflow.

B. Numerical analysis

Figure 3 depicts the rejection rate evolution for different values of system load. We selected physical graphs with 15, 30 and 50 nodes for this simulation. As the convergence time for the exact (ILP) approach increases exponentially with the graph size, the rejection rate is important for larger graph sizes (close to 80% in the worst case). When the load increases (i.e. the average number of new requests increases), the rejection rate also increases. Note that for the RL approach, the rejection rate is close to zero. This is due to the negligible necessary time to converge to near optimal solution using the Q-Learning approach (close to 0 in all considered scenarios).

To enlarge the grasp of the performance assessment of our approaches, Table III represents the results of a comparison between the Reinforcement Learning based approach and the exact approach using different metrics. Simulations

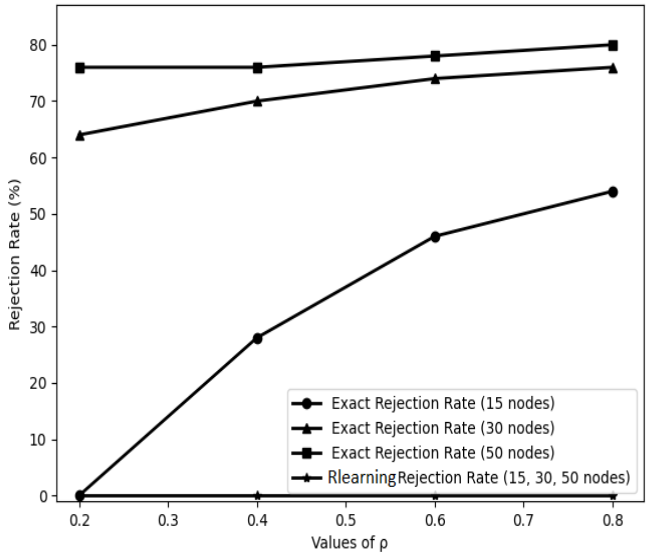

Fig. 3: Algorithms' Rejection rates for different graph sizes

correspond to graphs with a number of nodes in the $[5; 80]$ range. The exact algorithm still outperforms RL in terms of objective function metric (see columns 3 and 4 in Table III). Nevertheless we notice that the gap between the Exact and RL approaches has decreased for small and medium-size networks. Figure 4 confirms this result. However, for dense physical networks, RL approach needs more time in the training phase of the learning process, before converging to near optimal solutions. This is confirmed in Table III illustrating the worst case for larger graphs with a Gap close to 10%. To improve this weakness of the RL algorithm, we may estimate the Q-Values using Deep Reinforcement Learning (DQN)" (see [**?**], for instance).

TABLE II: Exact vs RL Algorithms Performance

# nodes	# links	Exa.Obj	RL.Obj	Exa.Time(ms)	RL.Time(ms)	Train Time RL(s)	Gap%
	7	18.35	17.11	14.72	0.921	1.61	6.79
	16	22.5	21.06	72.64	7.33	11.48	6.4
	25	24.42	22.59	171.72	26.18	17.07	7.49
	34	23.91	22.23	379.74	31.96	48.09	7.01
	43	24.77	22.92	632.81	54.02	93.99	7.44
	52	25.49	23.51	937.66	56.1	98.42	7.8
	70	25.96	23.6	14607.15	77.82	139.16	9.06
	88	26.17	23.67	26110.11	102.23	182.47	9.55
	124	26.4	23.75	58235.02	152.73	296.81	10.05
	142	26.81	24.12	91514.16	180.04	378.63	10

TABLE III: Exact vs RL Algorithms Performance

# nodes	# links	Exa.Obj	RL.Obj	Exa.Time(ms)	RL.Time(ms)	Gap%
	7	18.35	17.11	14.72	0.921	6.79
	16	22.5	21.06	72.64	7.33	6.4
	25	24.42	22.59	171.72	26.18	7.49
	34	23.91	22.23	379.74	31.96	7.01
	43	24.77	22.92	632.81	54.02	7.44
	52	25.49	23.51	937.66	56.1	7.8
	70	25.96	23.6	14607.15	77.82	9.06
	88	26.17	23.67	26110.11	102.23	9.55
	124	26.4	23.75	58235.02	152.73	10.05
	142	26.81	24.12	91514.16	180.04	10

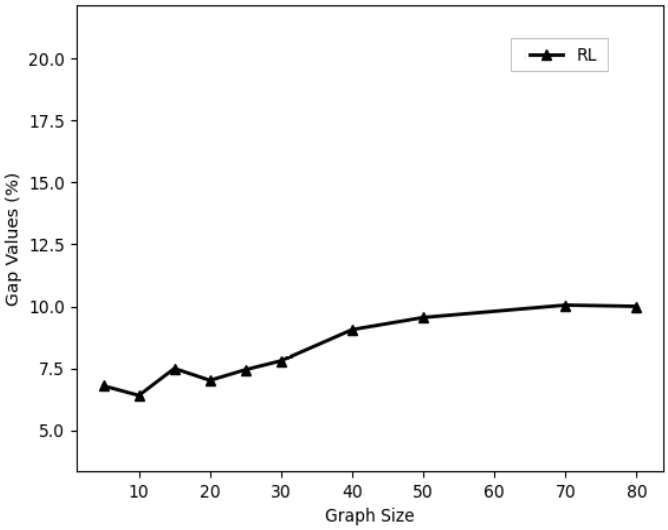

Fig. 4: RL-approach's gap evolution

V. Conclusion and future work

We addressed in this paper the virtualized face detection optimization problem when considering edge computing and networking architectures. Edge networking is able to bring the resources and intelligence close to connected devices. We focused on the mapping of virtual graphs representing face detection in Industry 4.0 on edge networks represented by graphs with limited resources. The considered optimization problem is a notary NP-Hard problem. Hence, we proposed an ILP approach to cope with small and medium problem instances, and a Reinforcement Learning solution is then proposed and deeply discussed to address scalability issues. We highlighted the efficiency of our approaches through simulations, using different graph requests and instances.

We showed the performance of our proposed solutions and illustrated their use-fullness in different scenarios considering face detection, and, we strongly believe their applications in other relevant Industry 4.0 use cases such as predictive maintenance and Fault detection.

For a future work, we propose to investigate new learning and distributed solutions based on Federated Learning (FL), which is a Machine Learning technique that enables us to train an algorithm across multiple decentralized edge devices or servers holding local data (see [12]). Our objective is to distribute the exact approach on smaller interconnected cameras/sensors and then use FL techniques to improve the scalability and the quality of the results.

References

[1] S. Jo and J. Chung, "Joint optimized cpu and networking control scheme for improved energy efficiency in video streaming on mobile devices," *Mobile Information Systems*, vol. 2017, 2017.

[2] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[3] S. Dräxler, H. Karl, and Z. Mann, "Jasper: Joint optimization of scaling, placement, and routing of virtual network services," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 946–960, 2018.

[4] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, and J. Henkel, "Computation offloading and resource allocation for low-power iot edge devices," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 7–12.

[5] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. [Online]. Available: http://arxiv.org/abs/1305.0209

[6] H. Moens and F. D. Turck, "Vnf-p: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 418–423.

[7] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 1346–1354.

[8] M.-I. Khaledi, M. Hadji, S.-E. El-Ayoubi, and D. Niyato, "Optimization of function chaining on the edge for iot applications," *IEEE WCNC, Beijing, China*, 2021.

[9] "https://www.etsi.org/technologies/689-network-functions-virtualisation," 2021.

[10] A. Ruiz-Garcia, V. Palade, I. Almakky, and M. Elshaw, "Deep q-learning for illumination and rotation invariant face detection," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[11] "https://www.ibm.com/fr-fr/analytics/cplex-optimizer," 2021.

[12] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," *CoRR*, vol. abs/1902.01046, 2019. [Online]. Available: http://arxiv.org/abs/1902.01046