Reinforcement Learning Framework for Server Placement and Workload Allocation in Multiaccess Edge Computing

Anahita Mazloomi, Hani Sami, Jamal Bentahar, *Member, IEEE*, Hadi Otrok, *Senior Member, IEEE*, and Azzam Mourad, *Senior Member, IEEE*

***Abstract*—Cloud computing is a reliable solution to provide distributed computation power. However, real-time response is still challenging regarding the enormous amount of data generated by the IoT devices in 5G and 6G networks. Thus, multiaccess edge computing (MEC), which consists of distributing the edge servers in the proximity of end users to have low latency besides the higher processing power, is increasingly becoming a vital factor for the success of modern applications. This article addresses the problem of minimizing both, the network delay, which is the main objective of MEC, and the number of edge servers to provide a MEC design with minimum cost. This MEC design consists of edge servers placement and base stations allocation, which makes it a joint combinatorial optimization problem (COP). Recently, reinforcement learning (RL) has shown promising results for COPs. However, modeling real-world problems using RL when the state and action spaces are large still needs investigation. We propose a novel RL framework with an efficient representation and modeling of the state space, action space, and the penalty function in the design of the underlying Markov decision process (MDP) for solving our problem. This modeling makes the temporal difference (TD) learning applicable for a large-scale real-world problem while minimizing the cost of network design. We introduce the TD(λ) with eligibility traces for minimizing the cost (TDMC) algorithm, in addition to *Q*-learning for the same problem (QMC) when $\lambda = 0$. Furthermore, we discuss the impact of state representation, action space, and penalty function on the convergence of each model. Extensive experiments using real-world data sets from Shanghai Telecommunication and Citywide Public Computer Centers demonstrate that in the light of an efficient model, TDMC/QMC are able to find the actions that are the source of lower delayed penalty. The reported results show that our algorithm outperforms the other benchmarks by creating a tradeoff among multiple objectives.**

***Index Terms*—Base station allocation, edge server placement, multiaccess edge computing (MEC), *Q*-learning, reinforcement learning (RL), TD(λ).**

Manuscript received 15 December 2021; revised 28 April 2022 and 3 August 2022; accepted 5 September 2022. Date of publication 8 September 2022; date of current version 6 January 2023. The work of Hani Sami was supported by FRQNT, Quebec. The work of Jamal Bentahar was supported in part by NSERC and DnD, Canada, and in part by FRQNT, Quebec. *(Corresponding author: Jamal Bentahar.)*

Anahita Mazloomi and Hani Sami are with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada (e-mail: ana_mazloomi@yahoo.com; hani.sami@mail.concordia.ca).

Jamal Bentahar is with the Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC H3G 1M8, Canada, and also with the Center of Cyber-Physical Systems, Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, UAE (e-mail: bentahar@ciise.concordia.ca).

Hadi Otrok is with the Center of Cyber-Physical Systems, Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, UAE (e-mail: hadi.otrok@ku.ac.ae).

Azzam Mourad is with the Cyber Security Systems and Applied AI Research Center, Department of CSM, Lebanese American University Division of Science, Lebanese American University, Beirut 10017, Lebanon, and also with the Division of Science, New York University Abu Dhabi, Abu Dhabi, UAE (e-mail: azzam.mourad@lau.edu.lb).

Digital Object Identifier 10.1109/JIOT.2022.3205051

I. Introduction

With the development of IoT [1] and 5G and 6G networks, applications of smart mobile devices in different areas, such as augmented reality, multiplayer games [2], image processing for facial recognition, natural language processing for real-time translation systems [3], and data transferring among IoT devices on the Internet of Vehicles (IoV) [4] have become more resource intensive. More computation power and real-time response are needed, but mobile devices are limited in terms of the central processing unit (CPU), memory, storage, and processing power.

First, mobile-cloud computing (MCC) [5], [6] was offered as a solution to overcome those constraints of handheld devices. Although cloud computing is a reliable solution to provide computation power, the real-time response is not guaranteed. The next solution was a cloudlet-based structure. Cloudlet is a group of powerful computers connected to the Internet [7]. The cloudlet has less computation power in comparison with the cloud, and because of the small coverage area, it is not scalable [8]. Finally, mobile-edge computing, currently known as multiaccess edge computing (MEC), was offered as a novel network to mitigate the cloudlet shortcomings. It brings the computation power close to the end users at the edge of the network, in a distributed manner [3]. Thus, besides having more process power, the delay is decreased because of the computation source proximity.

The mobile or IoT devices send a massive amount of requests to the edge servers, but there is a limited number of these servers because of the budget and energy consumption. Thus, the optimal locations for a limited number of edge servers among a large number of potential places should be found to have the minimum network delay [9]. Besides, the computation power of the edge servers is not unlimited. Hence, discovering the dominant area of each edge server is required as each of these servers can only handle a certain number

of requests [10]. Overloading results in more delay compared to sending the requests to a farther server [2]. Additionally, in assignments with idle edge servers, the network's cost increases, and the energy consumption of an idle edge server is about 60% higher than a fully loaded edge server [11]. Therefore, strategic placement and optimal base station assignment are necessary to have a high Quality of Service (QoS) and Quality of Experience (QoE) [12].

To solve the edge server placement problem, iterative/exhaustive search algorithms are space and time consuming. Due to the hardness of the problem and the possibility of having a large input size, the iterative search (such as K-means) is not guaranteed to produce efficient solutions in the case of multiobjective problems. Additionally, a heuristic-based solution is not guaranteed to reach an optimal solution even if the evolutionary-based algorithms are used.

Some studies have focused on task offloading by assuming that the edge servers' locations are known. In recent years, a number of them have considered the placement and task offloading together [13]. This joint problem is an NP-hard combinatorial optimization problem (COP) consisting of searching and finding the optimal option among a limited set of discrete possible solutions. Recently, reinforcement learning (RL) [14] has shown promising results for COPs, including traveling salesman to find the shortest path, mixed-integer linear programming, the graph coloring problem, and the Knapsack problem [15]. However, there are a few studies that have considered the use of RL for placement optimization [16], [17], [18], [19]. Although Q-learning is a powerful RL algorithm, in many real-world problems, finding the optimal solution is hard, especially when there is a large state or action space [15], [20], [21]. In addition, Q-learning suffers from overestimation bias in some cases. Thus, we expand our solution to consider n-step learning using TD(λ), where λ is the trace decay parameter [22]. If $\lambda = 0$ [i.e., TD(0)], the solution becomes the classical Q-learning. In this case, our RL solution for MEC design that considers the cost minimization is called QMC. If $\lambda \in (0, 1]$, we call the solution the TD(λ) framework for minimizing the cost (TDMC).

Considering the placement and task offloading joint problem, most proposals have assumed that the number of edge servers is fixed and the main objective is minimizing the delay. In this article, in addition to minimizing the delay, the main objective is to provide an efficient MEC design with minimum cost by minimizing the number of the edge servers. Minimizing the network delay along with the number of edge servers is a vital factor that should be considered toward having a competitive MEC design. This double objective should be attained under given constraints, including maximum accepted delay for the network and maximum workload capacity for edge servers. The output can be seen as different clusters of edge servers and their connected base stations. Providing an efficient RL framework for this capacitated clustering problem is our main focus. However, using RL for the server placement problem is highly challenging, and the number and distribution of the base stations, the delay and workload constraints, and the properties of edge servers are the factors that may increase the difficulty [20]. Moreover, applying RL in real problems is not trivial; efficient modeling is needed to overcome the RL limitations for the problems with large and variable action space [20], [23]. Furthermore, in our problem, there is no fixed situation as a goal for the agent, thus the reward value cannot be backtracked. In our MEC problem, every episode has a fixed set of steps where the agent must take a server placement decision for every base station. After a specific number of steps, the episode finishes without a final goal. This is different, for example, from the maze problem[1] where to find the optimal route there is a point when reached, it brings a high reward; otherwise, the length of the path is used as a reward. In this example, the agent can learn the optimal policy by backtracking the reward value. However, in our allocation problem, there is no end point with higher reward and the length of the path is the same in all episodes because the workload offloading should be done for all the base stations. When the agent reaches the base station n, the episode finishes, but it is not the final goal. Henceforth, selecting an optimal action systematically to achieve convergence is not always guaranteed.

Hence, the contributions of this article are as follows.

1) Providing a novel RL framework for the joint problem of edge server placement and workload allocation.
2) Defining the state and action spaces, and penalty function for multiobjective problems without a fixed end goal.
3) Adapting the temporal difference (TD) learning with a backward-view mechanism (TD(λ)) for a real-world and scalable problem with a large number of base stations (2798).
4) Our solution outperforms a deep version of RL that needs more computational resources.

We conducted extensive experiments on both the Shanghai Telecom and Citywide Public Computer Centers data sets. The results show that our solution creates a reasonable trade-off between our two objectives and outperforms other relevant benchmarks in the literature in terms of reducing the network delay and the number of edge servers. The benchmarks include the deep RL using deep Q-network (DQN), genetic algorithm (GA), K-means, Top-K, K-means Top-K merged (KmTK), TopDoF, and Random.

In Section II, after presenting an overview of RL, Q-learning, and backward view TD, we review the most relevant related work. In Section III, the system model and our formulation are reported. Then, in Section IV, our RL framework for edge server placement and computation offloading is introduced and explained in detail. The issues and limitations of Q-learning and TD(λ) implementations and the offered solutions are clarified. Section V shows extensive experiments, performance evaluations, and discussions after describing the real-world data set. Moreover, the other methods which are used for performance comparison are explained. Finally, in Section VI, a summary and some directions for future work conclude this article.

[1] https://www.samyzaf.com/ML/rl/qmaze.html

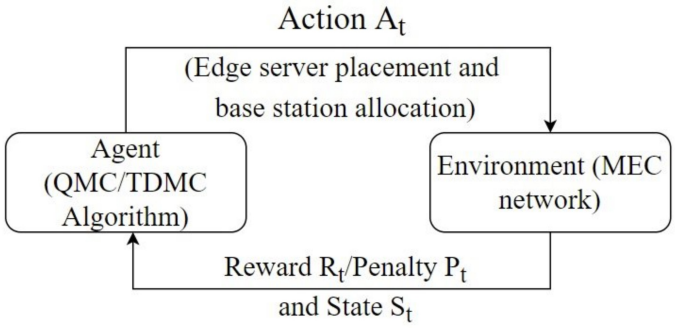


Fig. 1. RL framework.

II. Background and Related Work

Machine Learning has gained a great attention and success in the past few years for a broad range of domains [24], [25], [26]. RL is a type of machine learning with a learner called the agent. The agent makes decisions by interacting with its surrounding, i.e., the environment. The environment sends the reward or penalty signal and the new state to the agent after taking each action (see Fig. 1). Through this continuous process, the agent learns to map situations to the actions.

Generally, RL problems are formulated based on Markov Decision Processes (MDPs). Thus, in RL problems, state space $\mathcal{S}$, action space $\mathcal{A}$, reward $\mathcal{R}$, state transition probability $\mathcal{P}$, and discount factor γ should be defined through the MDP tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma, \mathcal{P} \rangle$.

Q-learning [27] is one of the first major RL algorithms, which is a model-free, off-policy algorithm defined by the following Bellman equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (1)$$

where $S_t, S_{t+1} \in \mathcal{S}$ are states at time t and $t+1$, $A_t \in \mathcal{A}$ is the action at time t, and $R_{t+1} \in \mathcal{R}$ is the immediate reward after taking each action. The Q-values represent the quality of actions in each state, and α is the learning rate that has a value between 0 and 1. These Q-values are stored in a tabular (Q-table) format where the rows and columns consist of the states and actions, respectively. Classic tabular learning updates involve using TD(0) (Q-learning) or TD(1) which is a Monte-Carlo implementation [22]. Despite the good performance entailed by each method, they suffer from bias and variance issues toward the action-value function (Q-value) update. To overcome this problem, we utilize TD(λ) for solving our problem. TD(0) involves a one-step Q-value update, while TD(1) waits until the end of the episode to update the Q-value of the traversed trajectory. A better solution is to consider an intermediate number of steps, which is specified by the *trace decay parameter* $\lambda \in [0, 1]$ and using the backward-view of TD(λ). In order to implement the backward-view, eligibility traces $e_t(s, a) \in \mathbb{R}^+$ is introduced for each state s and action a in a separate table. At each step, the eligibility trace for the visited state is increased by 1, while it decays for all the states by γ^λ as follows:

$$e_{t+1}(s, a) = \begin{cases} \gamma \lambda e_t(s, a), & \text{if } A_t = a, S_t = s \\ \gamma \lambda e_t(s, a) + 1, & \text{otherwise.} \end{cases} \quad (2)$$

Using traces, we are able to tell the degree of which a state is eligible to undergo learning changes based on the reinforcing event. The TD error performs proportional learning update to the visited states. Therefore, (1) is transformed into

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times \mathcal{E}_t e_t(S_t, A_t) \quad (3)$$

where $\mathcal{E}$ is the error term at time t. In this approach λ controls the tradeoff between bias and variance, and is guaranteed to influence the learning speed and the asymptotic performance in the function approximation.

By assuming the known locations for the servers in [28], Q-learning is used to find the optimal offloading policy where the objective is to reduce energy consumption with a fixed maximum delay as a constraint. After modeling the offloading as an MDP, the reward is calculated based on the time and energy consumption that should be minimized. Sen and Shen [29] also used Q-learning with three actions in each state. The arrival task can be processed on the edge, fog, or cloud. After the convergence of the Q-value, the Q-table is used for task allocation in edge servers. Huang *et al.* [30] and Zhao *et al.* [31] showed that their approach could reduce execution time and energy consumption. For the joint optimization of task offloading and bandwidth allocation, a DQN is used to learn the offloading policy where the objective is minimizing both the latency and energy consumption.

To define the optimal number of cloudlets to have a tradeoff between the QoS and the cost for the service provider, Peng *et al.* [32] used an improved affinity propagation algorithm. In their clustering, they considered user movement and load balancing. They divided the density of mobile users before and after moving into three clusters: sparse, discrete, and dense. The place of cloudlets would be changed by the density to cover more users. The authors generated a data set for different numbers of mobile users, and their algorithm outperformed K-means and minibatch K-means. Wang *et al.* [8] have proposed the first study for the edge server placement. They used mixed-integer programming (MIP) for edge servers placement and base station allocation by considering the access delay and load balancing as the two objectives for their formulation. It is considered that the fixed number of identical edge servers is given. Their results, on Shanghai Telecom's data set, are compared with the K-means, top-K, and random approaches for different numbers of base stations on a large scale, for 300 to 3000 base stations. Although their experiments showed K-means has less delay and top-K creates a more balanced cluster, in total, their approach has better results. In [33], for the same data set, they have used the MIP and K-means combination for 20 to 200 base stations. Xu *et al.* [34] used a three-step approach to find the edge server placement to minimize the delay while considering the workload balancing. They used a decision tree in the first level with the help of a GA and multiple criteria decision-making techniques in the following steps. They compared their proposed

approach with the two other greedy methods, where the nearest edge server or the edge server with the most available computing power is selected. To find the optimal placement and resource allocation in MEC, various approaches are investigated, including mainly: clustering [32], top-K [2], MIP [8], combination of MIP and K-means [33], heuristics [11], [34], ILP, and game theory [35].

There are a few studies that have considered RL in placement optimization. Mirhoseini *et al.* [17] proposed a deep RL algorithm to find the optimal placement for different operations of neural networks onto hardware devices that can be CPU or GPU. They used the square root of the execution time as a reward, and they demonstrated a reduction in the single-step running time as well as the total training time compared to the heuristics and traditional methods. Venkatakrishnan *et al.* [19] studied the same problem using the graph embedding method that helped generalize and improve the algorithm to be applicable for different neural networks, rather than only a specific one. The algorithm's output is a policy instead of places for operations. This learned policy is transferable for the unseen neural networks of the same family. In both [17], [19], before implementing the RL, the authors have grouped the same operations and forced the algorithm to place them in the same device, which reduces the placement actions. Mirhoseini *et al.* [16] used RL for chip placement. The performance of their algorithm improves over time by having more experience. As in [19], their algorithm can be employed for unseen blocks, even when these blocks have bigger size. However, unlike [19], in [16], the placement policy is defined incrementally after each state until reaching the terminal. The reward for each step is zero, except for the final one. To consider all the objectives, the authors used a weighted sum in a single reward function, which is computed by a neural network. This neural network is made by a rich state representation and a large data set to train it in a supervised manner. They have noticeably improved the placement time while outperforming other methods.

Applying RL for edge server placement still needs investigation. There is no fixed point to achieve as a goal and minimizing the path's penalty is the objective. Our agent needs to find the optimal nodes as edge servers that are fixed and do not vary over time. Therefore, in this article, a novel TD(λ)-based approach with a backward-view mechanism is examined and applied to design MEC with regard to optimized placement of edge servers while considering the deployment and delay cost as variables to be minimized.

III. Edge Server Placement Modeling

A. System Model and Problem Definition

MEC can be shown by a set of n base stations $BS = \{bs_1, bs_2, \ldots, bs_n\}$ and a set of K edge servers $ES = \{es_1, es_2, \ldots, es_K\}$, $(K < n)$. Edge server placement in 5G and 6G networks can be shown by an undirected graph $G = (V, E)$, where $V = BS \bigcup ES$ and E is the representation of the connections between base stations and edge servers. Usually, for simplicity purposes, it is assumed the edge servers are co-located with the existing base stations [8], [32].

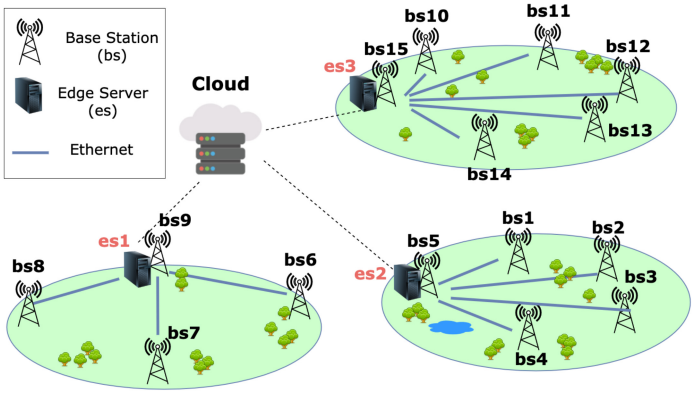


Fig. 2. Edge-server placement in MEC.

In Fig. 2, the bigger base stations illustrate the equipped base stations with servers. For example, $es1$ is the edge server that covers base stations $bs6$, $bs7$, $bs8$, and the co-located base station $bs9$. The workload of $es1$ is the total workload of all connected base stations. In order to have an optimal network design, the following assumptions are considered in our model.

1) Each edge server is co-located with one of the existing base stations.
2) Each edge server covers many base stations while each base station is only connected to one edge server, and the selected edge servers should cover all the base stations.
3) The distance between the base station and its edge server reflects the delay.

The MEC design refers to finding the optimal locations to add the servers, and base station allocation/workload offloading by using the historical data from users' connections to the base stations over days. These two problems are known to be NP-hard [8]; therefore, we are proposing an RL-based algorithm to solve them. The key variables used in our formulations and modeling are shown in Table I.

B. Computational Problem

Our objective is to minimize the network's cost, which relies on two parts: minimizing the number of edge servers K (4) and minimizing the network delay D_N (5). Deployment of new edge servers increases the cost of the network, and as mentioned having a minimum delay is one of the most vital objectives that should be considered. In fact, as argued in [3], the long execution time is one of the main concerns in the design of MEC. Therefore, this problem is modeled as follows:

$$O_1 : \min(K) \tag{4}$$

$$O_2 : \min(D_N) \tag{5}$$

subject to the following constraints:

$$\Delta_j \le \Delta_{\max} \quad (\forall j, 1 \le j \le K) \tag{6}$$

$$\sum_{j=1}^{K} \left(x_{ij} d_{ij}\right) \le D_{\mathrm{TH}} \quad (\forall i, 1 \le i \le n) \tag{7}$$

$$\sum_{j=1}^{K} x_{ij} = 1 \quad (\forall i, 1 \le i \le n) \tag{8}$$

$$x_{ij} \in \{0, 1\} \quad (\forall i, 1 \le i \le n \text{ and } \forall j, 1 \le j \le K). \tag{9}$$

TABLE I
NOTATIONS

Symbol	Meaning
BS	set of all the base stations in the network
bs_i	base station i in the network, $1 \leq i \leq n$
ES	set of all the edge servers in the network
es_j	edge server j in the network, $1 \leq j \leq K$
G	mobile edge computing network
K	number of edge servers
D_N	Network delay
n	number of base stations
$\lvert n_j \rvert$	number of base stations in cluster j
$\overline{D}$	set of average distances of all clusters
$\overline{D}_j$	average distances of base stations, in cluster j, from the j^{th} edge server
D_{TH}	distance threshold
d_{ij}	distance of base station i from edge server j
Δ	set of all edge servers' workloads in the network
Δ_j	workload of edge server j in the network
δ	set of all base stations' workloads in the network
δ_i	workload of base station i in the network
x_{ij}	binary variable, $x_{ij} = 1$ if base station i is connected to the edge server j
L_{bs_i}	location of the base station i
lat_{bs_i}	latitude of a base station i
lon_{bs_i}	longitude of a base station i
a_{bs_i}	action space of bs_i
α	learning rate
γ	discount factor
$ne1_{bs_i}$	1^{th} neighbor of bs_i with respect to the distance
wl_{ES}	dictionary of the edge servers' workloads
wl_j	workload of each edge server j, $1 \leq j \leq K$
z	binary variable if selected base station is a new edge server $z = 1$
Pr_i	priority of bs_i

The first constraint (6) states each edge server has a limited computational power. An upper bound $\Delta_{\max}$ is considered for the edge servers as the processing capability. As the distance reflects the delay, D_{TH} is used to represent the distance threshold representing the maximum acceptable delay (7). It is worth noting that, although an acceptable constraint for the latency is considered (7), as mentioned above in (5), still our goal is always to further minimize the delay. Knowing that $D_{N_i,j} = ([d_{i,j}]/[\text{Speed}_{i,j}])$, the delay is derived from the distance, where $D_{N_i,j}$ and $\text{Speed}_{i,j}$ are the networking delay and propagation speed between i and j, respectively.

Finally, each base station is connected to just one edge server while all the base stations are covered (8), x_{ij} is a binary variable (9) and $x_{ij} = 1$ if base station i is connected to the edge server j; otherwise, $x_{ij} = 0$.

IV. Algorithmic Implementation

The joint problem of finding the optimal placement and workload offloading is modeled as an RL problem. Moving forward from one base station to the other one, from 1 to n, is the path in each episode. At each time step, the agent is in the location of one of the base stations. Then, the agent takes action for the base station and receives the result in the form of a penalty. This procedure continues until the agent moves through all the base stations. After enough iterations and experiencing different interactions with the environment, the agent could be able to find the actions with the minimum penalty for the path.

A. Initial State Space

As RL is based on MDP, defining the state space, action space, and the reward function are the fundamental steps. First, the state space is defined and the agent moves forward from one base station to the next one. Thus, the state space is a set of locations of all base stations because the agent locates in the place of one of them at each time step. This location has two geographical coordinations: 1) latitude and 2) longitude. The initial state and the location are formalized as follows:

$$\text{Initial\_State\_set} = \left[L_{bs_1},\ L_{bs_2}, \ldots,\ L_{bs_n} \right]$$
$$L_{bs_i} = \left[\text{lat}_{bs_i},\ \text{lon}_{bs_i} \right].$$

B. Action Space

Second, by considering the predefined distance threshold based on the delay budget, the action space is defined. We create an adjacency matrix of zeros and ones for our network. It is a square matrix where rows and columns show the base stations. In this symmetric matrix, the entries are zero unless their value in the distance matrix is less than the threshold. The distance matrix has the same dimension, but the values show the distance between the adjacent base stations. If there are n base stations, this matrix will be $n \times n$. Then, the Hadamard product of each row of the adjacency matrix and the joint action is the neighbor vector for each of the base stations. The neighbor vector shows the possible action space for each base station. Finally, the nonzero elements are sorted based on the distance matrix and the result is the base stations action space. Therefore, the action space for each base station is limited to the nodes with value one in the adjacency matrix (i.e., the neighbors). The joint action space of all base stations is

$$A = \{a_{bs_1}, a_{bs_2}, \ldots, a_{bs_n}\}$$
$$a_{bs_i} = [ne1_{bs_i}, ne2_{bs_i}, \ldots, ne_{N_{bs_i}}]$$

where a_{bs_i} shows the action space of base station i which is limited to its neighbors and $ne1_{bs_i}$ is the first neighbor of base station i. The size of this action space varies for each node, and it is equal to the number of the neighbors (N) of each base station. For instance, assuming there are five base stations and the distance matrix is as follows:

$$\begin{bmatrix} 0 & 5 & 8 & 9 & 4 \\ 5 & 0 & 3 & 17 & 2 \\ 8 & 3 & 0 & 10 & 14 \\ 9 & 17 & 10 & 0 & 20 \\ 4 & 2 & 14 & 20 & 0 \end{bmatrix}.$$

In case $D_{\text{th}} = 7$, the adjacency matrix becomes

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Afterward, the neighbor vector for each base station is computed based on the Hadamard product of the first row in the adjacency matrix and the joint action space

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} bs_1 & bs_2 & bs_3 & bs_4 & bs_5 \end{bmatrix} = \begin{bmatrix} bs_1 & bs_2 & 0 & 0 & bs_5 \end{bmatrix}.$$

Finally, the set of feasible actions for bs_1 becomes

$$a_{bs_1} = [bs_1, bs_5, bs_2].$$

Variable action space is a challenging situation in deep RL [36], [37], [38], [39]. This issue arises when function approximation should be used. Hence, it is preferred to model the problem in a way that can be solved by a Q-table.

C. Initial Penalty Function

Next, the environmental signal, which is sent to the agent as the penalty (P), is determined. This signal leads the agent toward taking the optimal actions in the given states. The objective is minimizing the cost, which consists of delay and the number of base stations.

We align the standard Q-learning formula to our objective function as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times \left[P_{t+1} + \gamma \min_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right]. \quad (10)$$

P_{t+1} represents the immediate penalty, and the agent's policy is to minimize the penalty, instead of maximizing the reward. We considered $\alpha = 0.4$ and $\gamma = 0.9$ for our algorithm.

Goldie and Mirhoseini [20] asserted that penalty/reward calculation for the placement problems when the action space is not completed (partial placement) is very difficult. Furthermore, for multiobjective problems, it is usually hard to determine a single penalty value. To evaluate the policy, it is crucial to consider the performance of the algorithm for each objective separately [23].

D. Challenges and Solutions

Distance, as the first factor, was used in calculating the penalty because it reflects the delay in our modeling. By having this penalty function, after iterations, the action in each base station (state) is choosing itself as the destination. In this situation, the distance is zero, which minimizes the penalty. This result is in contrast with the other objective (4). Therefore, a balance between the two goals is needed.

To address the above issue, a fixed value, after having a new edge server, is added to the penalty. Setting this value is challenging and should be defined based on the model's constraints. In our algorithm, it should be higher than the distance threshold. Otherwise, the agent would add extra edge servers to the network while it could select one of the existing ones in the accepted distance. For example, when our distance threshold was 9 km, this fixed value was considered 10. This motivated the agent to select existing edge servers within 9 km of the current base station rather than adding a new edge server because of receiving less penalty.

Although adding this value helps the agent take better actions, it is not enough for our algorithm to converge to the minimum cost. It is because of the actions' dependency and the lack of a static final goal. For example, as mentioned earlier, to find the optimal route in the maze problem,[2] there is a point when reached, it brings a high reward; otherwise, the length of the path is used as a reward. Then, the agent can learn the optimal policy by backtracking the reward value. However, in our allocation problem, there is no end point with higher reward and the length of the path is the same in all episodes because the workload offloading should be done for all the base stations. When the agent reaches the base station n, the episode finishes, but it is not the final goal. Therefore, the agent is not able to find a systematic rule to select the optimal action based on Q-values changes and thus, convergence is not guaranteed.

Moreover, as the length of the path increases, the agent faces the credit assignment problem (CAP) [40]. The agent is not able to define which action is resulting in a higher penalty because, in our problem, the lower penalty in the current state does not guarantee the minimum cost (penalty) for the completed path or even in the next episode.

1) Modified State Space: To address the two aforementioned issues: 1) lack of end goal and 2) CAP, a list of previously taken actions in each episode is added to the state definition at each time step. In other words, the list of edge servers up to the current time step is appended to the state space as follows:

$$\text{Modified\_State}_t = [L_{bs_t}, [a_{s_1}, a_{s_2}, \ldots, a_{s_{t-1}}]].$$

For completing the penalty function after a precise definition of state and action space, constraints should be considered. Penalizing the actions that do not satisfy the constraints with a high penalty is a possible solution. In this case, the agent could be trapped, if there are numerous infeasible actions. Additionally, there might be a probability of not satisfying the constraints, and it needs more time and more iterations to reach the optimal value. The other method that can be applied is to prevent the agent from taking forbidden actions by setting some rules. In our study, the latter approach is chosen.

2) State Space: For the distance constraint, as explained previously, the action space is limited to the neighbors. For the workload constraints, a dictionary of the current edge servers and their workloads is added to the state definition (wl_{ES}), which gets updated after each action. Therefore, our complete state definition at time step t is:

$$\text{Final\_Modified\_State}_t = [L_{bs_t}, [a_{s_1}, a_{s_2}, \ldots, a_{s_{t-1}}] \{es_1{:}wl_1, es_2{:}wl_2, \ldots\}].$$

This definition simplifies the penalty calculation, shrinks the action space further, and keeps the agent away from taking infeasible actions.

3) QTable: There is another situation that the agent should avoid. For instance, if the agent in bs_5 selects bs_{10} as the destination for offloading, it cannot map the other base stations to bs_5. For this reason, a list of forbidden actions (FA) is created. As the agent moves forward, the list expands, and the action space becomes smaller, but it causes variable action space for each state in each episode, which, as mentioned earlier, is a highly challenging problem for a DQN. Besides, the following reasons make DQN unsuitable for our problem: 1) each input with a different size might require a new network design;

[2]https://www.samyzaf.com/ML/rl/qmaze.html

2) DQN requires a large number of hyperparameters to be tuned for each configuration, such as minibatch size, replay buffer size, neural network configuration, optimizer configuration, loss function configuration, etc.; and 3) our problem does not require a nonlinear approximation because the states do not change over time. As shown in the evaluation section (Section V-D), the DQN solution is not capable of converging to the minimum cost compared to our proposed TD(λ) solution.

Our modeling solves the aforementioned difficulties, namely: large and variable action space, penalty calculation, CAP, and lack of exact end goal. However, to use the Q-table, the large state space challenge has to be addressed. In the Q-table, the states are the rows, and the actions are columns. In our state definition, there is a fixed part, the base station location, which is the same in all the episodes. The other components vary based on different actions. In our Q-table, as the final goal is to have the best action for each base station, only the fixed part is considered as the state to be fitted in the look-up table (Algorithm 1 line 6). It should be noted that in different episodes, the agent uses the variable parts, the selected edge servers and their workloads, to calculate the penalty and take the appropriate actions. Thus, the variable parts function as a memory for our agent in the decision-making process because the fixed section is not able to represent the whole information of each state. Therefore, the $n \times n$ matrix is created as our Q-table. Initially, it is filled out with a high value of 1000. Then, the entries with value one in the adjacency matrix change to zero (Algorithm 1, line 1). These initial action values motivate the agent for exploration, and the possible actions would be tried several times before the convergence [14].

4) Penalty Function: Having completed the model, the remaining issue is that the actions are dependent on the node orders. For example, the agent always selects the first node as an edge server. If the order of nodes changes, the edge servers will change as well. To tackle this problem, a priority value Pr is created for each node. It can partially show the importance of nodes for being selected as the edge server. For example, selecting a node with a higher workload is preferred because it reduces the workload transferring, which results in less delay. Also, if a node is in a dense area, it is a more suitable place for adding the edge servers. The priority Pr_i of the base station i is expressed as follows:

$$\mathrm{Pr}_i = (\delta_i + \mathrm{DoF}_i)/\mathrm{average\_distance}_i;\ (\forall i,\ 1 \le i \le n) \tag{11}$$

where δ_i is the workload of base station i. DoF_i is the degree of freedom that shows the number of neighbors situated in the accepted distance. *DoF* represents the directions that each base station can transfer its computation or receive tasks from other nodes. $\mathrm{average\_distance_i}$ is the average distance of $N(=15)$ nearest nodes from the base station i. The inverse of the Pr expression is the penalty of selecting each node, which helps the agent make decisions regarding the importance of the nodes not based on their orders.

Therefore, the penalty function is

$$P = \mathrm{distance}\left(L_{bs_i}, L_{a_{bs_i}}\right) + \mathrm{fixed\_value} \times z + 1/\mathrm{Pr}_{a_{bs_i}} \tag{12}$$

where the distance is calculated based on (14)–(18), which will be explained in Section V. The input of the distance function, $\mathrm{distance}(L_{bs_i}, L_{a_{bs_i}})$, is the location of the current node, L_{bs_i}, and the location of the selected node for computation offloading. The action of the current state is also a base station, and its location ($L_{a_{bs_i}}$) should be used in (12). The fixed_value is added if the selected node has not been in the list of the edge servers (ES) which is the reason for considering the z as a binary variable. The last part is the inverse of the priority of the selected destination.

E. Initialization and Exploration

So far, the fundamental parts for creating the efficient TD learning algorithm are explained. An additional step is needed to make this efficient modeling applicable to our problem. In each episode, a list of possible actions (PA) for each state is created. Initially, the list is empty, and as the agent passes the states, it fills based on the actions. For example, in each state, this list is a subset of the current edge servers that are within the current state's acceptable distance. Then, based on the ϵ-greedy policy, the agent in each state selects the actions from the possible-action list or the action space that was defined for the classic Q-learning algorithm. The parameter ϵ, which represents the probability of selecting a random action, is calculated as follows:

$$\epsilon = 9/(T + 100) \tag{13}$$

where T is the number of episodes. In the first iterations, the action selection is rather based on exploration. After some iterations, the agent relies on exploiting the best-offered actions based on the Q-value. Consequently, the possible-action list, based on this policy, is improved as well.

F. TDMC Algorithm

Algorithm 1 represents our proposed TDMC, and as mentioned earlier, when $\lambda = 0$, the algorithm becomes QMC (Q-learning for minimizing the cost). RESETENVIRONMENT() refers to the process of updating the edge servers list in each episode (ES = [ ]), updating the workload dictionary of the selected edge servers ($wl_{\mathrm{ES}} = \{\ \}$) and updating the forbidden action's list (FA = [ ]). Lines 6 and 7 force the agent to follow the mentioned rule that a node cannot be both sender and receiver.

In Fig. 3, we present a flowchart of Algorithm 1 for more detailed illustration of the TDMC training process.

TDMC starts by initializing the Q-values and eligibility trace values to 1000 and 0, respectively. For every episode, the environment is reset before iterating over the number of base stations. TDMC keeps track of the list of edge servers. If the base station being evaluated is an edge server, the workload and rest of parameters are updated. If the current base station is not an edge server, the list of possible actions is evaluated. If there exist available actions, the action having the minimum Q-value is selected based on the ϵ-greedy algorithm. In case there is no possible action, the neighbors are retrieved based on the distance threshold, and the ϵ-greedy algorithm is evaluated to select an action. Finally, the parameters including the

Algorithm 1: TDMC Framework

Initialize $Q(s, a)$ to 1000 except adjacent nodes for each state to 0, and $Q(terminal, .) = 0$
Initialize $e(s, a)$ to 0
for *episode* $\leftarrow$ *1 to MAX* **do**
    RESETENVIRONMENT()
    for $i \leftarrow 1$ *to* n **do**
        $s_i \leftarrow L_{bs_i}$
        if s_i *in ES* **then**
            $a_{bs_i} \leftarrow bs_i$, obtain P, $s_{i+1} \leftarrow L_{bs_{i+1}}$, update Q, update e, update ES, update wl_{ES}
        else
            $PA \leftarrow$ GETPOSSIBLEACTIONS()
            $Q\_PA \leftarrow$ GETQVALUESOFPA()
            if $|PA| > 0$ **then**
                Choose a_{bs_i} for s_i using ϵ-greedy policy from Q_PA
            else
                $Nei \leftarrow$ GETNEIGHBORS()
                $Q\_Nei \leftarrow$ GETQVALUESOFNEI()
                Choose a_{bs_i} for s_i using ϵ-greedy policy from Q_Nei
            take action a_{bs_i}, obtain P, $s_{i+1} \leftarrow L_{bs_{i+1}}$, update Q, update e, UPDATEENVIRONMENT()
            `/* UpdateEnvironment(): update ES, update` wl_{ES} `and update FA */`

workload of the edge servers, actions, penalty, Q-value, and eligibility traces are updated.

V. RESULTS AND DISCUSSION

To evaluate the performance of our proposed algorithms, a data set from Shanghai Telecommunication [8], [33], [41], [42] in China is used. It comprises the information related to 3233 base stations and the connected users over 30 days, in June 2014. The number of requests is used for workload calculation. The number of all the requests from different users that are directed to each base station every day represents the base station's workload. After computing this value for each of the base stations over 30 days, the maximum value is the base station's workload, $\delta = \{\delta_1, \delta_2, \ldots, \delta_n\}$. Thus, in the workload calculation, the worst situation is considered because the maximum workloads may not happen concurrently. The reason is that the placement of selected nodes as edge servers does not change over a short period of time. Moreover, because of the fast growth of the number of mobile and IoT devices, this conservative workload calculation is preferred. Then, based on the capacity limitation of the edge servers, the maximum workload that each edge server can handle was considered 150 requests per day similarly to the setup in [43]. Considering the base stations' workload distribution, some of them with higher workloads were omitted from our data set.

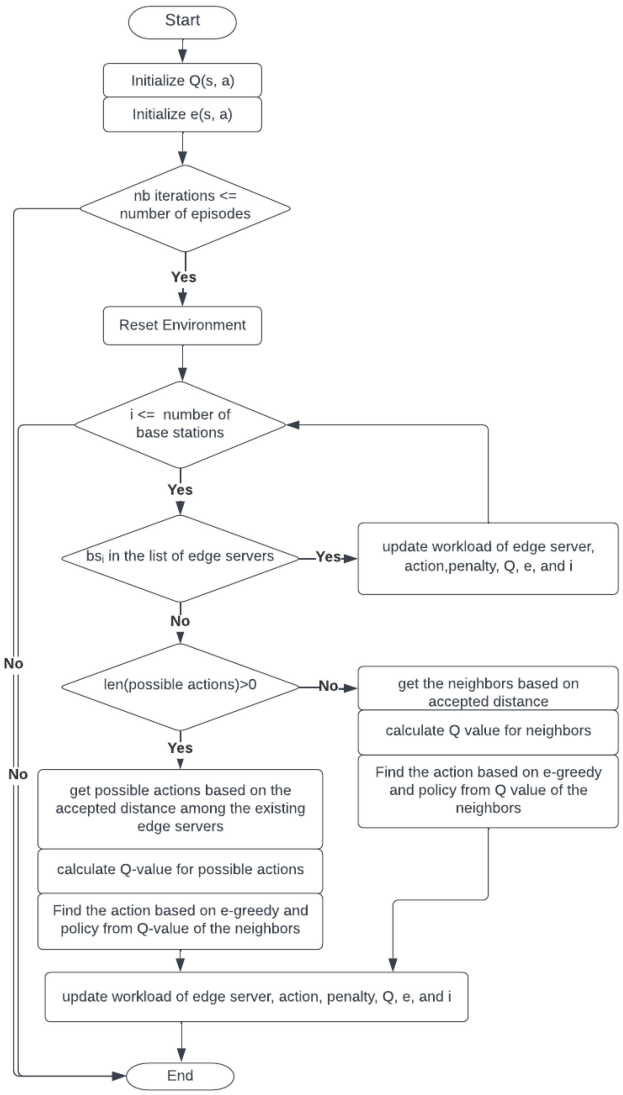


Fig. 3. TDMC flowchart.

Furthermore, we evaluate the performance of QMC and TDMC on the Citywide Public Computer Centers data set.[3] We provide the evaluation results of this data set in Appendix A. In the Appendix, we also provide additional experiments on the use of different λ values in TDMC, the impact on initializing the Q-value to different numbers, and the runtime of QMC and TDMC for different numbers of base stations.

As mentioned, the distance represents the delay. The distance matrix is created using the geographical locations in the data set. For that, the fourth column of the data set is divided into two separate columns. Instead of the Euclidean distance that is the length of a straight line between two points, the following equations are used to have more precise distances:

$$d\text{lon} = \text{lon2} - \text{lon1} \tag{14}$$

$$d\text{lat} = \text{lat2} - \text{lat1} \tag{15}$$

[3]https://data.cityofnewyork.us/Social-Services/Citywide-Public-Computer-Centers/cuzb-dmcd

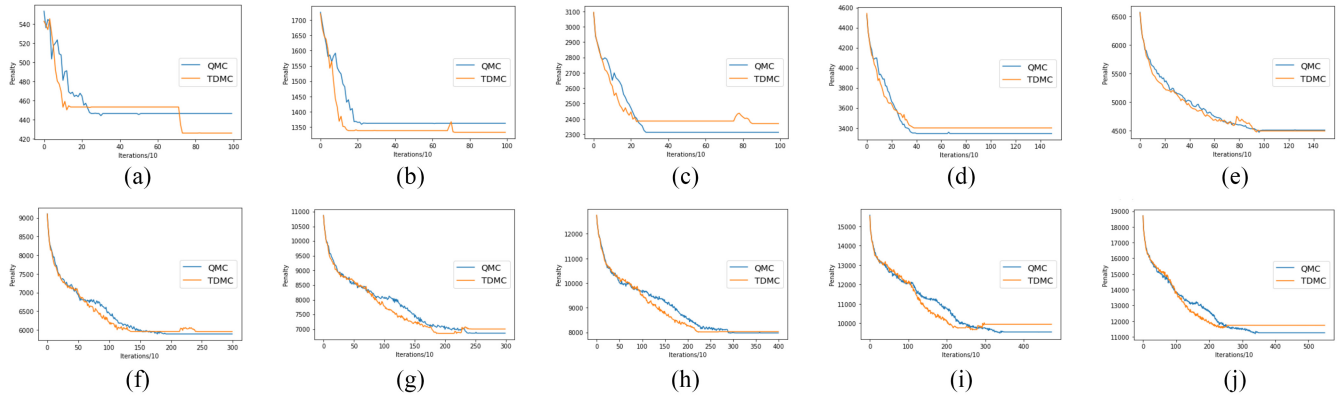


Fig. 4. Implementing QMC and TDMC to find minimum edge servers for different numbers of base stations. (a) 100 base stations. (b) 300 base stations. (c) 500 base stations. (d) 700 base stations. (e) 1000 base stations. (f) 1400 base stations. (g) 1700 base stations. (h) 2000 base stations. (i) 2400 base stations. (j) 2798 base stations.

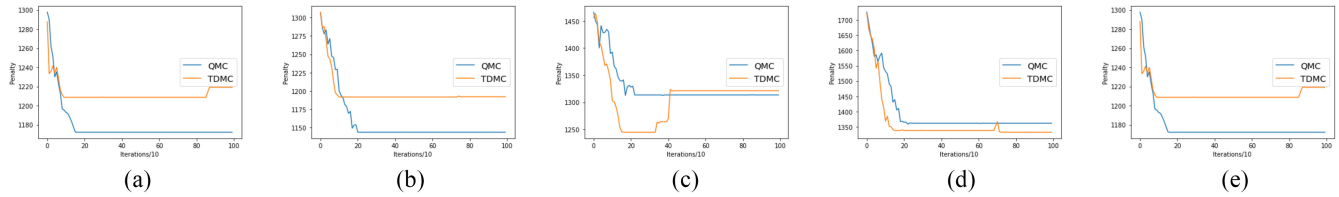


Fig. 5. Performance of our proposed algorithm by considering different accepted network delays. (a) Distance threshold = 3 km. (b) Distance threshold = 5 km. (c) Distance threshold = 7 km. (d) Distance threshold = 9 km. (e) Distance threshold = 11 km.

$$a = (\sin(d\text{lat}/2))^2 + \cos(\text{lat1}) \times \cos(\text{lat2}) \times (\sin(d\text{lon}/2))^2 \tag{16}$$

$$c = 2 \times a\tan2(sqrt(a), sqrt(1-a)) \tag{17}$$

$$d = R \times c \tag{18}$$

where lon and lat represent the longitude and latitude of each node in radians, respectively. Equation (16) is the haversine formula that computes the great-circle distance between two points in (17) [44]. This distance is the shortest path between the two points on the surface of the sphere. The $a\tan2(x, y)$ is the arctangent that gives the radians angle between x and y. R is the radius of the earth that is equal to 6371 km. In the last step, the base stations which have fewer than five neighbors in their 3-km distance are known as the outliers and are removed from the data set.

In our constrained MDP, two constraints of maximum workload and maximum distance need to be defined before implementation. The maximum capacity of each edge server is considered equal to 150 requests per day, and the maximum acceptable delay for our network is 0.03 ms. To satisfy the delay constraint, the distance between the edge server and the dominated base stations should be within 9 km [43].

A. Implementation Results

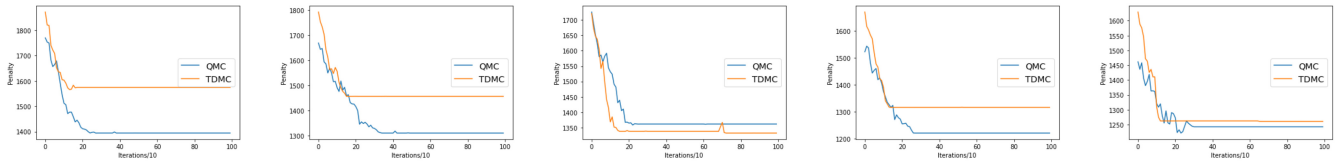
Based on our proposed algorithms, namely, QMC and TDMC, the agent could find the near-optimal actions for different numbers of base stations, and the convergence is guaranteed after enough iterations (Fig. 4). The cost, the y-axis, represents the completed path's penalty in each episode. As illustrated, the performances of our two algorithms are relatively similar. However, when the number of base stations becomes big (> 300), QMC shows slightly better results and can reach less penalty in comparison with TDMC ($\lambda = 0.4$). Fluctuations in the cost value in the first iterations show the learning process of the agent through trial and error. When the number of base stations increases, the action space expands, and consequently, more iterations are needed to reach the optimal actions.

Although only the fixed part of our state definition is used in creating the Q-table, the state-action values have converged. Interestingly, our algorithms worked for a large number of base stations [Fig. 4(f)–(j)] with the Q-table without the help of neural networks by keeping a part of the state definition as the memory in each episode.

Fig. 5 demonstrates the convergence of our algorithms while the distance threshold varies from 3 to 11 km. It means the maximum accepted latency varies between 0.01 and 0.036 ms. This experiment is done for 300 base stations, and the maximum workload is considered 150 requests per day. By increasing the distance threshold, more iterations are needed because the action space is increasing, as well. In the other experiment, the convergence of the model is shown for 300 base stations and the maximum acceptable delay of 0.03 ms, while the workload's limit changes between 100 to 200 requests per day (Fig. 6). Increasing the workload capacity does not necessarily increase the convergence time [comparing Fig. 6(b) and (c)], and is dependent on the node's position and the number of neighbors. These two experiments (Figs. 5 and 6) show that as the distance threshold increases, the performances of two algorithms become more similar.

B. Performance Evaluation

1) TDMC Versus QMC With Varying Constraints: It is supposed that the processing time is negligible, and by following



(a) (b) (c) (d) (e)

Fig. 6. Performance of our proposed algorithm by considering different computation capacities for the edge servers. (a) Computation capacity = 100 requests/day. (b) Computation capacity = 120 requests/day. (c) Computation capacity = 150 requests/day. (d) Computation capacity = 170 requests/day. (e) Computation capacity = 200 requests/day.

the workload condition, there is no queue, and consequently, there is no waiting time. Thus, the access delay represents the network delay. Therefore, to evaluate the performance of different algorithms, first, the average distance that represents the communication or access delay and the number of selected edge servers will be reported. Then, the combination cost which is the summation of total delay and the cost of adding new edge servers will be analyzed to give us a better view while considering the both objective functions (4) and (5). Before going to the results, a brief explanation of the algorithms is given.

In Top-K, the nodes based on their workloads are sorted, and the first one, with the highest workload, is selected as the edge server. Then, the neighbors of the selected edge server are defined. It includes all the nodes that their distance is less than the predefined threshold. The neighbor nodes offload their computation to the elected edge server until reaching the maximum capacity of the server. This cluster of the edge server and its connected base stations is removed from the initial set of base stations. Then, the remaining nodes are again sorted based on their workloads, and the algorithm is repeated until all the base stations are assigned to an edge server. Top-DoF and Random algorithms, have the same structure but the edge servers' selecting criteria are different. In Top-DoF, the nodes are sorted based on the number of their neighbors. The node with the most neighbors is selected as the edge server. In the Random algorithm, the first node as the edge server is picked randomly. In both, then the adjacent nodes (neighbors) are mapped to the selected node until reaching the maximum accepted workload. This procedure continues till all nodes are allocated to an edge server. K-means, where K should be defined in advance, is one of the most common unsupervised algorithms. As the objective of our solution is to find the number of edge servers, i.e., K, we implemented K-means repetitively by increasing the parameter K until satisfying the load and distance constraints. The other algorithm is the combination of K-means and Top-K (KmTK). In this method, K-means is executed first until meeting the distance constraint in each cluster. Then, the node having the highest load is selected as the head of each cluster (edge server), before running the Top-K at the end. Finally, in the GA, we implemented the solution that generates random populations and applies mutation and crossover with the same input, cost function, and output that we use in TDMC and QMC.

2) Network Delay, Edge Servers, and Cost With Varying Number of Base Stations: Tables II and III illustrate the performance of the considered algorithms for different numbers of base stations ranging from 100 to 2798. Each table represents one of the objectives. The network delay for repetitive K-means is zero in most cases. On the other hand, the number of edge servers is close to the number of base stations, which is similar to the situation when each base station is selected as the edge server. The combination of K-means and Top-K (KmTK) has reduced the network delay compared to the outcomes of Top-K. However, the number of edge servers has increased. KmTK has less delay compared to QMC and TDMC, but it has more number of edge servers with a significant difference. The performance of the Random is interesting, and in most cases, it has some striking similarities to the performance of the Top-K in terms of network delay. Considering the number of edge servers, GA has the minimum cost while the network delay is higher than some other methods. Therefore, by comparing these two tables, creating a combination cost is crucial to have a holistic view

TABLE II
NETWORK DELAY (KM) WITH RESPECT TO THE NUMBER OF BASE STATIONS

n	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
100	2.74	2.46	**0.1**	3.8	1.97	2.87	4.44	3.47
300	2.48	2.42	**0.1**	4	2.89	3.22	4.64	4.26
500	2.33	2.48	**0**	4.11	2.47	3.33	4.27	4.15
700	2.26	2.41	**0**	4.16	2.34	3.53	4.55	4.29
1000	2.21	2.2	**0**	4.32	2.38	3.4	4.49	4.36
1400	2.13	2.19	**0**	4.4	2.02	3.82	4.47	4.36
1800	2.14	2.23	**0**	4.42	1.96	3.9	4.52	4.5
2000	2.16	2.2	**0**	4.55	2.04	3.96	4.49	4.66
2400	2.18	2.31	**0**	5.09	1.91	4.14	4.56	5.02
2798	2.19	1.36	**0**	5.59	1.11	4.17	4.58	5.42

TABLE III
NUMBER OF SELECTED EDGE SERVERS WITH RESPECT TO THE NUMBER OF BASE STATIONS

n	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
100	**17**	18	69	19	29	18	18	18
300	59	59	208	63	93	**57**	63	64
500	109	109	469	114	226	**107**	112	114
700	171	171	672	176	342	**170**	172	175
1000	**224**	225	968	230	470	**224**	228	230
1400	285	284	1346	290	621	**278**	288	292
1800	318	318	1697	323	706	**312**	324	328
2000	360	358	1895	363	851	**352**	364	365
2400	428	428	2282	433	1000	**422**	435	443
2798	509	510	2662	512	1216	**501**	517	521

$$\text{combination\_cost} = \sum_{i=1}^{n} \sum_{j=1}^{K} d_{ij} + 10 \times K. \tag{19}$$

In (19), K is the number of the edge servers and as the maximum accepted delay is 9 km, the cost of adding each edge

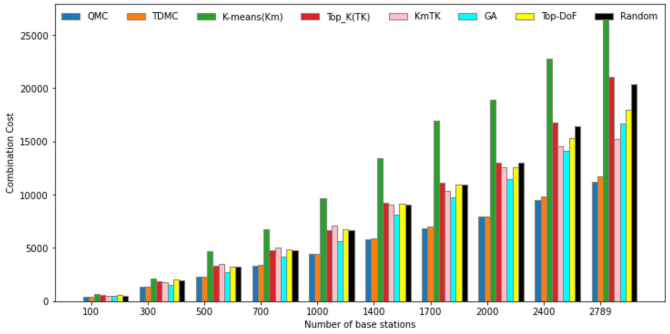


Fig. 7. Cost comparison by considering the two objectives for different numbers of base stations.

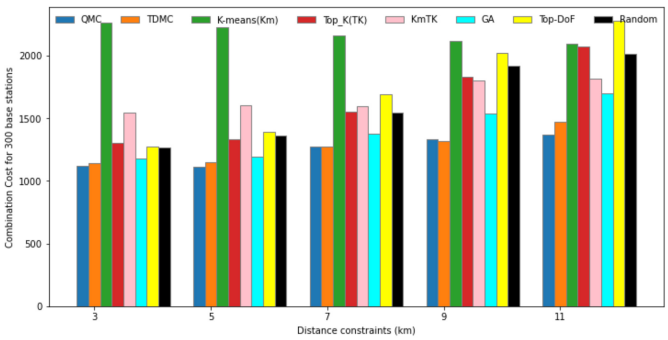


Fig. 8. Cost comparison by considering the two objectives for different distance constraints.

TABLE IV
NETWORK DELAY (KM) WITH RESPECT TO THE DISTANCE CONSTRAINT FOR 300 BASE STATIONS

D_{TH}	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
3	1.1	1.07	**0.076**	1.21	0.71	1.25	1.31	1.18
5	1.48	1.6	**0.077**	1.98	1.74	1.88	2.17	2.14
7	2.18	2.22	**0.093**	2.94	1.92	2.59	3.34	3.01
9	2.48	2.42	**0.108**	4	2.89	3.22	4.64	4.26
11	2.65	3	**0.112**	4.87	3.01	3.8	5.55	4.65

TABLE V
NUMBER OF SELECTED EDGE SERVERS WITH RESPECT TO THE DISTANCE CONSTRAINT FOR 300 BASE STATIONS

D_{TH}	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
3	**79**	82	224	94	133	80	88	91
5	67	67	220	74	108	**63**	74	72
7	62	61	213	67	102	**60**	69	64
9	59	59	208	63	93	**57**	63	64
11	57	57	206	61	91	**56**	61	62

server is considered equal to 10. Fig. 7 shows the combination cost of all the algorithms for different numbers of base stations. The combination cost of QMC and TDMC is less than the other algorithms, while for 100 and 300 base stations, TDMC performs batter. In the other cases, TDMC and QMC are very close. After these two algorithms, GA shows in most of the cases better performance compared to the rest of the algorithms because both objectives are considered simultaneously. Up to 2000 base stations, KmTK, Top-K, Top-DoF, and Random reveal very close outcomes. When the number of base stations increases, KmTK shows better cost followed by Top-Dof. Top-K and Random show similar trends. Finally, K-means is ranked last.

3) Network Delay, Edge Servers, and Cost With Varying Distance: In the other experiment, the number of base stations is 300, and the maximum computation capacity of edge servers is 150 requests per day (Tables IV and V). These two factors are fixed, and the distance constraint varies. The performance of algorithms is compared in terms of the number of edge servers (K) in Table V, and in terms of network delay (D_N) in Table IV, which represents the quality of the network.

Less delay, as an integral feature of the MEC, results in higher QoS and customer satisfaction. It is clear that increasing the distance constraint results in higher latency for all the algorithms. In most of the cases, after K-means, our QMC and TDMC algorithms have less delay (Table IV).

TABLE VI
NETWORK DELAY (KM) WITH RESPECT TO THE EDGE SERVERS COMPUTATION CAPACITY FOR 300 BASE STATIONS

Δ_{TH}	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
100	1.46	2.17	**0.01**	3.2	1.97	3.13	3.55	3.22
120	1.87	2.28	**0.07**	3.37	2.46	3.24	3.77	3.12
150	2.48	2.42	**0.108**	3.39	2.89	3.22	4.02	3.16
170	2.17	2.57	**0.18**	3.36	2.93	3.36	3.91	3.55
200	2.47	2.74	**0.29**	3.56	2.84	3.43	4.2	3.42

The number of selected edge servers decreases as the delay increases, except in one case for the Random algorithm. The Random algorithm has exceptional results compared to Top-K and Top-DoF, and it outperforms in some situations. Similar to the previous experience, GA has reached the minimum number of edge servers except for 3-km distance threshold, where QMC has fewer edge servers (Table V).

As shown in Fig. 8, QMC and TDMC have the lowest combination cost for different distances. Then, GA followed by Top-K show better cost compared to the rest of algorithms for small distances. As the distance constraint becomes less tight, KmTK outperforms Top-K, but GA still reveals better cost. We can also observe that the cost of K-means decreases while increasing the distance to the point that this cost becomes close to the one generated by Top-K and less than the cost of Top-DoF when $D_{\text{th}} = 11$ km. In fact, although Top-DoF shows competitive cost for small distances, the approach becomes expensive for long distances.

4) Network Delay, Edge Server, and Cost With Varying Workload: In our last experiment, the effect of workload constraint is investigated. Similar to the previous one, 300 base stations are considered when the maximum accepted delay is 0.03 ms, but here different edge servers capacities are examined.

Concerning the network delay, after K-means, QMC noticeably outperforms the others benchmark approaches (Table VI). The TDMC and then KmTK algorithms, on average, have a better performance compared to Random, Top-K and Top-DoF. To find the minimum number of edge servers, as the capacity increases, the number of edge servers decreases. GA has the best performance for finding the smallest number of edge servers, then TDMC outperforms the other algorithms (Table VII). As the tables show, for each objective, different algorithms have better performance compared to the others, so

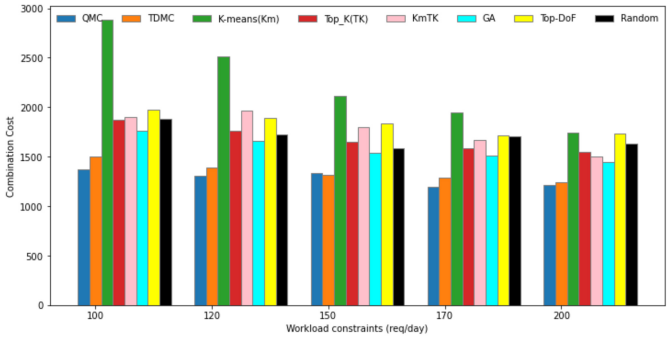


Fig. 9. Cost comparison by considering the two objectives for different edge servers computation capacities.

TABLE VII
NUMBER OF SELECTED EDGE SERVERS WITH RESPECT TO THE EDGE SERVERS COMPUTATION CAPACITY FOR 300 BASE STATIONS

Δ_{TH}	QMC	TDMC	K-means (Km)	Top-K (TK)	KmTK	GA	Top-DoF	Random
100	93	85	288	91	131	**82**	91	92
120	75	71	249	75	123	**69**	76	79
150	59	59	208	63	93	**57**	63	64
170	54	52	189	58	79	**50**	54	64
200	47	42	166	48	65	**42**	47	61

TABLE VIII
IMPACT OF α AND γ ON THE PERFORMANCE OF THE AGENT. $n = 700$; $D_{\text{TH}} = 5$ KM; AND 120 REQUEST/DAY

α	γ	Cost
0.9	0.5	2982.8254
0.4	**0.5**	**2982.8254**
0.4	0.9	2967.9766
0.2	0.9	2972.5821
0.2	0.2	2982.8254
0.7	0.7	2982.8254
0.7	0.9	2982.8254
0.8	**0.3**	**2967.9766**

we need to compare the combination cost of these algorithms. Fig. 9 reveals that in all the cases, QMC and TDMC outperform the other solutions, with a slightly better advantage for QMC. The third best algorithm is GA followed by Random and Top-K that, on average, have better performance compared to KmTK and Top-K, while in all cases, the K-means has the highest cost.

Based on all these experiments, it can be concluded that QMC, TDMC, and GA have less cost because these algorithms consider both objectives: 1) network delay and 2) number of edge servers simultaneously.

C. Selection of Parameters α and γ

In our experiments, deciding on the values of α and γ is critical for the agent performance in terms of quality of the decisions. In Table VIII, we show the minimum cost achieved by the agent while varying α and γ. To this end, we consider 700 base stations and set the maximum distance to be 5 km with 120 requests per day as a workload.

Based on the results shown in Table VIII, with higher values of α, the agent takes bigger steps in the learning process with a probability of missing some better actions, which could result in lower costs. When α is too small, it takes more iterations to converge and the results also might not be satisfying. The magnitude of γ shows the importance of future rewards. Both of these hyperparameters should be defined based on each problem/input size. In addition, we can see that the minimum cost is obtained in two settings. Therefore, our decision for choosing $\alpha = 0.4$ and $\gamma = 0.9$ is based on trial and error for different values of these two parameters.

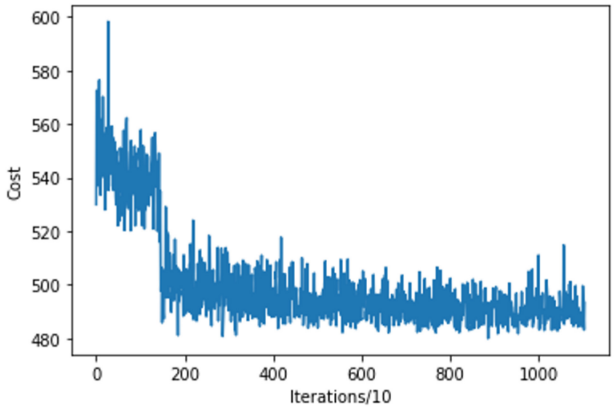


Fig. 10. DQN performance with 100 base stations.

D. Deep Reinforcement Learning

In this experiment, we evaluate the performance of DQN to solve our edge server placement problem, while considering the same constraints and MDP design. The performance of the DQN agent is shown in Fig. 10. In this figure, we can notice that DQN is not able to converge to a smaller cost or stabilize the decisions compared to QMC and TDMC [see Fig. 4(a)]. The main reason behind the poor performance of DQN is that our efficient MDP design does not require a nonlinear approximation that DQN performs to approximate the Q-table. Implementing this nonnecessary approximation causes the model to generate and select nonoptimal actions, which results in unstable cost values. Moreover, DQN requires a high number of parameters that require further training and careful tuning.

VI. CONCLUSION

In this article, a new RL-based algorithm was developed for jointly optimizing the placement and computation offloading to minimize the cost of the MEC design. The challenges of conducting this research are the limitations of RL when used for real-world applications, including the consideration of variable action space, difficulty in penalty definition for multiobjective problems, lack of a specific static end goal, and the curse of dimensionality caused by massive action space and state space explosion. Our RL-based frameworks provide an efficient solution for this joint problem while considering these issues. Despite the mentioned challenges, the key to Q-learning application for this large-scale real-world problem is a precise and efficient representation of the state space, action space, and penalty function. In addition to obtaining a convergent solution for our proposed algorithms, which is a difficult challenge in RL, we have proposed an efficient formalization that

resulted in promising outcomes. The performance obtained from our algorithms demonstrated that the RL-based solutions have better performance in cost reduction, which includes both reducing network delay and the number of the edge servers, compared to relevant benchmark methods. We also showed that applying a deep reinforcement solution resulted in a poor outcome since the MDP design does not require a nonlinear approximation. For future work, we are investigating the ability to generalize the algorithm for larger and unseen networks by defining the MEC as a graph and then deploying a graph representation to encode the state and action spaces.

APPENDIX A
ADDITIONAL EXPERIMENTS

In this Appendix, we evaluate the performance of TDMC on a new data set named Citywide Public Computer Centers, which offers descriptions for more than 400 computing machines with beneficial properties covering the physical location and workloads. Mapping these computing machines to base stations in the context of edge server placement, we are able to perform our experiments. This Appendix includes cost evaluations compared to QMC and GA, the impact of varying λ in TDMC, the impact of initializing the Q-values randomly, and a study of runtime.

A. Performance Evaluation

In this section, we present the results for evaluating the QMC and TDMC performance compared to GA for various numbers of base stations. The results are presented in Fig. 11. As revealed by the figure, the performance of TDMC in terms of the convergence cost is the lowest among QMC and GA for all the different numbers of base stations. Therefore, this again illustrates the advantage of the proposed MDP formulation and TDMC solution on the Citywide Public Computer Centers data set.

Even though the running time of the TDMC is significantly higher than QMC and GA, we can see that the outcome of TDMC is significantly better. In Appendixes A–D, we study the runtime of TDMC and QMC for different numbers of base stations.

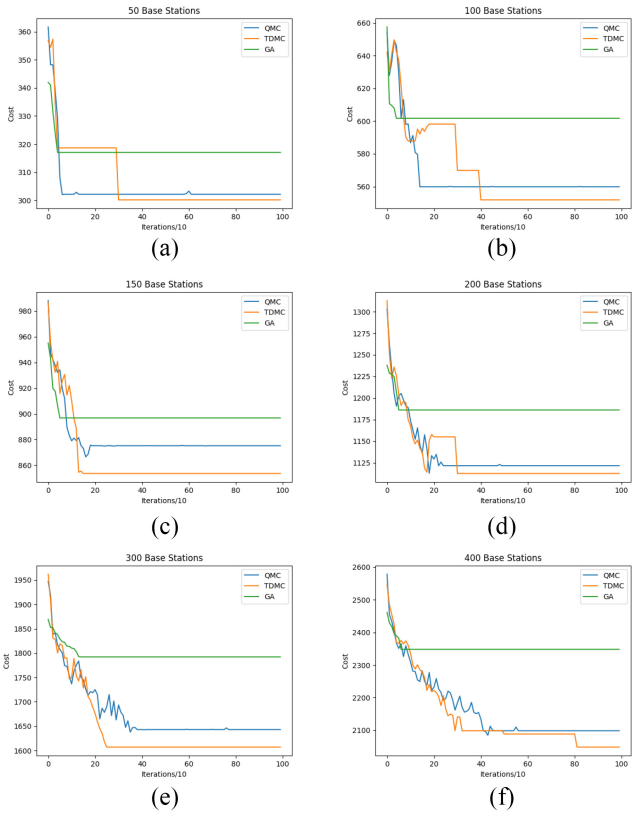


Fig. 11. Performance evaluation of TDMC and QMC compared to GA for the Citywide Public Computer Centers data set. (a) 50 base stations. (b) 100 base stations. (c) 150 base stations. (d) 200 base stations. (e) 300 base stations. (f) 400 base stations.

B. Varying λ

Changing the value of λ affects the performance of TDMC for different numbers of base stations and various data sets. Based on the results shown in Fig. 12, the TDMC achieves the best performance when $\lambda = 0.4$ with the lowest cost compared to 0.2, 0.3, 0.5, 0.6, 0,7, and 0.8. The performance of TDMC in this data set is similar to the performance using the Shanghai Telecommunication data set, where TDMC has the best overall performance.

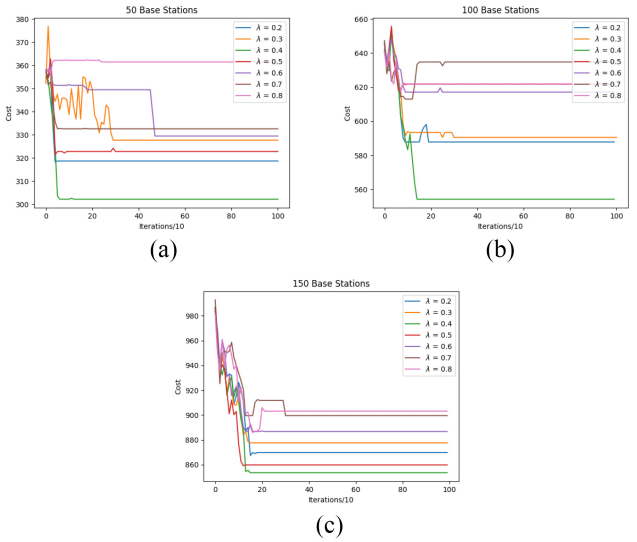


Fig. 12. Performance of TDMC for different λ. (a) 50 base stations. (b) 100 base stations. (c) 150 base stations.

C. Q-Value Initialization

In this section, we study the impact of initializing the Q-values in the Q-table of TDMC to different values. More specifically, we evaluate the performance of TDMC when the Q-values are initialized to 0, 1000, and random values between 0 and 1000. As illustrated in Fig. 13, TDMC achieves the best performance with the lowest cost when initialized to 1000. Compared to QMC for 100 base stations, we can see that QMC is never able to reach below 560 as a cost compared to TDMC. This again shows the advantage of the proposed TDMC solution for $\lambda = 0.4$.

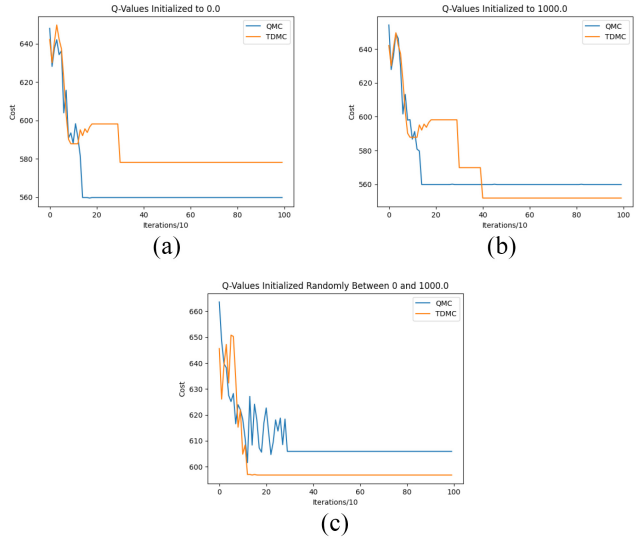


Fig. 13. Performance of TDMC for different Q-values initialization. (a) Initializing Q-value to 0. (b) Initializing Q-value to 1000. (c) Initializing Q-value randomly between 0 and 1000.

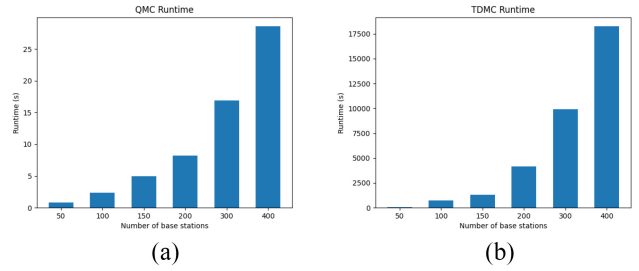


Fig. 14. Runtime of (a) QMC and (b) TDMC for different numbers of base stations.

D. Runtime of QMC and TDMC

Our problem formulation for edge placement on base stations requires a single run of TDMC on the given data set. Our solution offers a design and assignment of edge servers based on the locations of base stations and average expected workload. We assume in this work that the workload is fixed. Thus, there is no need for running TDMC multiple times to adapt to the change of the workload. Henceforth, it is preferable to use TDMC to reach better near optimal solutions even though the runtime increases compared to other solutions.

In this section, we present the runtime of QMC and TDMC for different numbers of base stations. Our implementation is using the CPU (without GPU) on a Core i7 machine with 32 GB of RAM. In the bar graphs of Fig. 14, we can notice that QMC is much faster compared to TDMC. The increase in execution time for TDMC is caused by the fact that the eligibility trace grid should be updated for all states and actions at each iteration. As mentioned earlier, getting faster results is not the case with our problem settings. This is because the design of edge servers on physically fixed positions of base stations is done once per data set. Updating the edge servers periodically is part of our future work, which could depend on other factors, such as the change in workload over time and the flexibility of placing edge servers using the containerization technology.

REFERENCES

[1] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[2] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet al.ocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Oct.–Dec. 2017.

[3] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[4] A. Hammoud, H. Sami, A. Mourad, H. Otrok, R. Mizouni, and J. Bentahar, "AI, blockchain, and vehicular edge computing for smart and secure IoV: Challenges and directions," *IEEE Internet Things Mag.*, vol. 3, no. 2, pp. 68–73, Jun. 2020.

[5] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, 2012, pp. 21–28. [Online]. Available: https://doi.org/10.1145/2307849.2307856

[6] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Netw. Appl.*, vol. 16, no. 3, pp. 270–284, 2011. [Online]. Available: https://doi.org/10.1007/s11036-011-0305-7

[7] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.

[8] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, May 2019.

[9] X. Zhang, Z. Li, C. Lai, and J. Zhang, "Joint edge server placement and service placement in mobile edge computing," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 11261–11274, Jul. 2022.

[10] Y. Chen, Y. Lin, Z. Zheng, P. Yu, J. Shen, and M. Guo, "Preference-aware edge server placement in the Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1289–1299, Jan. 2022.

[11] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, 2018, pp. 66–73.

[12] Y. Li, A. Zhou, X. Ma, and S. Wang, "Profit-aware edge server placement," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 55–67, Jan. 2022.

[13] J. Meng, C. Zeng, H. Tan, Z. Li, B. Li, and X. Li, "Joint heterogeneous server placement and application configuration in edge computing," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2019, pp. 488–497.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] N. Mazyavkina, S. I. Sviridov, S. V. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," 2020, *arXiv:2003.03600*.

[16] A. Mirhoseini *et al.*, "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*.

[17] A. Mirhoseini *et al.*, "Device placement optimization with reinforcement learning," 2017, *arXiv:1706.04972*.

[18] K. E. Murray and V. Betz, "Adaptive FPGA placement optimization via reinforcement learning," in *Proc. ACM/IEEE 1st Workshop Mach. Learn. CAD (MLCAD)*, 2019, pp. 1–6.

[19] S. B. Venkatakrishnan, S. Gupta, H. Mao, M. Alizadeh, and R. Addank, "Learning generalizable device placement algorithms for distributed machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 3981–3991.

[20] A. Goldie and A. Mirhoseini, "Placement optimization with deep reinforcement learning," in *Proc. Int. Symp. Phys. Design*, 2020, pp. 3–7.

[21] Y. Chen *et al.*, "Reinforcement learning meets wireless networks: A layering perspective," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 85–111, Jan. 2021.

[22] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *Proc. ICML*, 2001, pp. 417–424.

[23] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019, *arXiv:1904.12901*.

[24] S. Fu, W. Liu, K. Zhang, Y. Zhou, and D. Tao, "Semi-supervised classification by graph p-Laplacian convolutional networks," *Inf. Sci.*, vol. 560, pp. 92–106, Jun. 2021.

[25] H. Sami, J. Bentahar, A. Mourad, H. Otrok, and E. Damiani, "Graph convolutional recurrent networks for reward shaping in reinforcement learning," *Inf. Sci.*, vol. 608, pp. 63–80, Aug. 2022.
[26] Y. Wu, R. Wang, M. Gong, J. Cheng, Z. Yu, and D. Tao, "Adversarial UV-transformation texture estimation for 3D face aging," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 7, pp. 4338–4350, Jul. 2022.
[27] C. J. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[28] B. Dab, N. Aitsaadi, and R. Langar, "*Q*-learning algorithm for joint computation offloading and resource allocation in edge cloud," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, 2019, pp. 45–52.
[29] T. Sen and H. Shen, "Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems," in *Proc. IEEE 3rd Int. Conf. Fog Edge Comput. (ICFEC)*, 2019, pp. 1–10.
[30] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, 2019.
[31] R. Zhao, X. Wang, J. Xia, and L. Fan, "Deep reinforcement learning based mobile edge computing for intelligent Internet of Things," *Phys. Commun.*, vol. 43, Dec. 2020, Art. no. 101184.
[32] K. Peng, X. Qian, B. Zhao, K. Zhang, and Y. Liu, "A new cloudlet placement method based on affinity propagation for cyber-physical-social systems in wireless metropolitan area networks," *IEEE Access*, vol. 8, pp. 34313–34325, 2020.
[33] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Softw. Pract. Exp.*, vol. 50, no. 5, pp. 489–502, 2020.
[34] X. Xu *et al.*, "Load-aware edge server placement for mobile edge computing in 5G networks," in *Proc. 17th Int. Conf. Service-Oriented Comput.*, 2019, pp. 494–507.
[35] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 494–503, Jan. 2021.
[36] C. Boutilier *et al.*, "Planning and learning with stochastic action sets," 2018, *arXiv:1805.02363*.
[37] H. Sami, H. Otrok, J. Bentahar, and A. Mourad, "AI-based resource provisioning of IoE services in 6G: A deep reinforcement learning approach," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3527–3540, Sep. 2021.
[38] Y. Chandak, G. Theocharous, B. Metevier, and P. S. Thomas, "Reinforcement learning when all actions are not always available," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 3381–3388.
[39] H. Sami, A. Mourad, H. Otrok, and J. Bentahar, "Demand-driven deep reinforcement learning for scalable fog and service placement," *IEEE Trans. Services Comput.*, early access, Apr. 27, 2021, doi: 10.1109/TSC.2021.3075988.
[40] M. Minsky, "Steps toward artificial intelligence," *Proc. IRE*, vol. 49, no. 1, pp. 8–30, 1961.
[41] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "QoS prediction for service recommendations in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 134–144, May 2019.
[42] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
[43] S. Lee, S. Lee, and M. K. Shin, "Low cost MEC server placement and association in 5G networks," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2019, pp. 879–882.
[44] R. W. Sinnott, "Virtues of the haversine," *Sky Telescope*, vol. 68, no. 2, p. 158, 1984.

Anahita Mazloomi received the B.S. degree in mechanical engineering from Yazd University, Yazd, Iran, in 2010, the first M.Sc. degree in industrial engineering from the Kharazmi University (Tehran Campus), Tehran, Iran, in 2013, and the second M.Sc. degree in information systems engineering from Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada, in 2021.

Her research topics are edge computing and reinforcement learning.

Hani Sami received the B.S. degree in computer science from Lebanese American University, Beirut, Lebanon, in 2017, and the M.Sc. degree in computer science from the American University of Beirut, Beirut, in 2019. He is currently pursuing the Ph.D. degree in information systems engineering with Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada.

His research topics are fog computing, vehicular fog computing, and reinforcement learning.

Mr. Sami is a reviewer of several prestigious conferences and journals.

Jamal Bentahar (Member, IEEE) received the Ph.D. degree in computer science and software engineering from Laval University, Québec, QC, Canada, in 2005.

He is a Professor with Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, Canada. From 2005 to 2006, he was a Postdoctoral Fellow with Laval University, and then an NSERC Postdoctoral Fellow with Simon Fraser University, Burnaby, BC, Canada. He is a Visiting Professor with Khalifa University of Science and Technology, Abu Dhabi, UAE. His research interests include the areas of computational logics, reinforcement learning, multiagent systems, service computing, game theory, and software engineering.

Prof. Bentahar was an NSERC Co-Chair for Discovery Grant for Computer Science from 2016 to 2018.

Hadi Otrok (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, QC, Canada, in 2008.

He holds a Full Professor position with the Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, UAE, an Affiliate Associate Professor with the Concordia Institute for Information Systems Engineering, Concordia University, and the Electrical Department, Ecole de Technologie Superieure, Montreal. His research interests include: blockchain, reinforcement learning, federated learning, crowdsensing and sourcing, ad hoc networks, and cloud and fog security.

Prof. Otrok is an Associate Editor of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *Ad-Hoc Networks* (Elsevier), and IEEE TRANSACTIONS ON SERVICES COMPUTING.

Azzam Mourad (Senior Member, IEEE) received the M.Sc. degree in computer science from Laval University, Québec, QC, Canada, in 2003, and the Ph.D. degree in electrical and computer engineering from Concordia University, Montreal, QC, Canada, in 2008.

He is currently Professor of Computer Science and the Founding Director of the Cyber Security Systems and Applied AI Research Center, Lebanese American University, Beirut, Lebanon, a Visiting Professor of Computer Science with New York University Abu Dhabi, Abu Dhabi, UAE, and an Affiliate Professor with the Software Engineering and IT Department, Ecole de Technologie Superieure, Montreal. His research interests include cybersecurity, federated machine learning, network and service optimization and management targeting IoT and IoV, cloud/fog/edge computing, and vehicular and mobile networks.

Prof. Mourad has served/serves as an Associate Editor for the IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE NETWORK, IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY, *IET Quantum Communication*, and IEEE COMMUNICATIONS LETTERS, the General Chair for IWCMC2020, the General Co-Chair for WiMob2016, and the track chair, TPC member, and reviewer for several prestigious journals and conferences.