REINFORCEMENT LEARNING FRAMEWORKS FOR SERVER PLACEMENT IN MULTI-ACCESS EDGE COMPUTING

Anahita Mazloomi

A thesis
in
The Department
of
Concordia Institute of Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science in Quality Systems Engineering
Concordia University
Montréal, Québec, Canada

December 2020

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Anahita Mazloomi**

Entitled: **Reinforcement Learning Frameworks for Server Placement in Multi-Access Edge Computing**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Quality Systems Engineering

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Roch Glitho _____ Chair

Dr. Olga Ormandjieva _____ Examiner

Dr. Rachida Dssouli _____ Examiner

Dr. Jamal Bentahar _____ Supervisor

Approved _____

Dr. Mohammad Mannan Graduate Program Director

_____ 2020 _____

Dean of Gina Cody School of Engineering and Computer Science

Abstract

Reinforcement Learning Frameworks for Server Placement in Multi-Access Edge Computing

Anahita Mazloomi

In the IoT era and with the advent of 5G networks, an enormous amount of data is generated, and new applications require more and more computation power and real-time response. Although cloud computing is a reliable solution to provide computation power, the real-time response is not guaranteed. Thus, the multi-access 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.

Edge server placement and task offloading play a crucial role in the efficient design of MEC architecture. There is a finite discrete set of possible solutions, and finding the optimal one is known to be an NP-hard combinatorial optimization problem. Heuristics, mixed-integer programming, and clustering algorithms are among the most widely used approaches to solve this problem.

Recently, researchers have investigated reinforcement learning (RL) to solve combinatorial optimization problems, which has shown promising results. In this thesis, we propose novel RL-frameworks for solving the joint problem of edge server placement and base station allocation. There are a few studies that have used RL in placement optimization. In our investigation, the focus is on the modeling part to make the Q-learning applicable for a large scale real-world problem.

Therefore, in this research, Q-learning is examined and applied in the edge server placement while considering two significant and striking perspectives. The first one is about minimizing the cost of network design by reducing the delay and the number of edge servers. The second perspective is the placement of K-edge servers to create K-fair-balanced clusters with minimum network delay.

Despite the impressive results of RL, its application in real-world scenarios is highly challenging. Throughout our modeling, the faced issues are explained, and

our solutions are provided. Besides, the impact of state representation, action space, and penalty function on the convergence is discussed. Extensive experiments using a real-world dataset from Shanghai demonstrate that in the light of efficient penalty function, the agent is able to find the actions that are the source of higher delayed rewards, and our proposed algorithms outperform the other benchmarks by creating a trade-off among multiple objectives.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Jamal Bentahar, for allowing me to pursue my passion while gently guiding me with his constructive comments and valuable advice through each stage of this study. Thank you for your amazing support during my whole graduate program.

I would like to extend my gratitude to my lovely parents and my family for their continuous support and encouragement. I also want to give a special thanks to my dear uncle for his warm support during these years. You are all a constant source of inspiration.

Last but not least, a bunch of thanks to my beloved husband for accompanying me through this journey with his endless love.

Thank you all!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter begins with presenting the context of this research, which is about server placement in multi-access edge computing networks. The motivations behind using reinforcement learning (RL) for the edge server placement and base station allocation are then explained. After stating the main objectives and based on the discussed challenges, our contributions are stated. Finally, the outline of this thesis is given.

1.1 Context

Over recent years, the popularity of smart mobile devices has noticeably increased due to their applications in different areas, such as reading news, entertainment, social networks, and learning [1]. These applications with the development of IoT and 5G networks have changed and become resource-intensive. For instance, more computation power and real-time response are needed for applications such as augmented reality, multiplayer games [19], image processing for facial recognition, natural language processing for real-time translation systems [2] and data transferring among IoT devices in the internet of vehicles (IoV) [16]. However, mobile devices are limited in terms of central processing unit (CPU), memory, storage, and processing power.

First, mobile cloud computing (MCC) [4, 53] was offered as a solution to overcome those constraints of handheld devices. In MCC, heavy tasks are offloaded to the remote cloud-based data centers with more computation resources. This centralized infrastructure receives all the data or requests from different users and provides high processing power, CPU, and storage [37]. Task offloading to the remote cloud results

in significant energy savings and better battery life for mobile devices [52, 21, 2].

Although cloud computing is a reliable solution to provide computation power, the real-time response is not guaranteed. By sending requests to the cloud, as it is typically far from the users, the network experiences more delay that is not desirable [19]. Furthermore, by the development of IoT and 5G networks, the amount of produced data by a large number of mobile devices increases dramatically. Therefore, the cloud will be subjected to a tremendous amount of data that increases the delay further. Hence, the limitations of this centralized solution arise the need for a new structure. We need a network that can handle numerous requests in real-time.

The next solution was a cloudlet-based structure. In [40], the authors have described the cloudlet as a group of powerful computers that are connected to the internet. In their proposed architecture, a mobile user by using virtual machine technology can offload tasks to the nearby cloudlet over a Wi-Fi access point. The cloudlet has less computation power in comparison with the cloud. Moreover, because of the small coverage areas, it is not scalable [49]. A detailed comparison of the cloud-based structure and cloudlet-based structure is given in [41].

Finally, the mobile edge computing, currently known as multi-access edge computing (MEC) network 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 [2]. Thus, besides having more process power, the delay is decreased because of the computation source proximity. In the MEC, the coverage area increases to the range of radio access network (RAN), and the MEC nodes or servers are usually co-located with the base stations. More details are provided in [11] where the authors have compared different edge computing implementations, including cloudlet and MEC.

1.2 Motivation

MEC is attracting considerable interest due to real-time response for the computation-intensive tasks. The edge servers play a significant role in the MEC architecture. The mobile or IoT devices send a massive amount of requests to the edge servers, but there is a limited number of edge 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. Moreover, 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. It seems to select the closest base stations for each edge server has the minimum delay for the network, but it can overload the edge servers. Overloading results in more delay, compared to sending the requests to a farther server [19]. Additionally, this assignment may result in having idle edge servers, which increases the cost of the network. The energy consumption of an idle edge server is about 60% higher than a fully-loaded edge server [23]. Therefore, strategic placement and optimal base station assignment are necessary to have high quality of service (QoS) and quality of experience (QoE).

Several studies have investigated MEC due to its growing importance, for instance, for smart cities applications. Most of these proposals have focused on task offloading assuming the edge servers' locations are known. In recent years, some researchers have studied the edge server placement problem, but a number of them have considered the placement and task offloading together [28]. It is worth noting that the place of servers is not always known in advance. Thus, the edge server placement problem is about finding the optimal places to add the edge servers. As it is assumed that servers are co-located with the base stations, it can be seen as selecting a limited number of base stations as the location for the edge servers. The optimal locations are the nodes that guarantee, after defining their dominant area, that the access/communication delay is minimized. Thus, in the placement optimization, the task offloading needs to be considered as well. The task/computation offloading is the process of sending the users requests received by the base stations to a server to be computed. It is assumed that each base station can offload its computations to a specific edge server while a server is covering many base stations. Therefore, discovering the optimal clusters of servers and connected base stations is crucial in order to have a minimum access delay besides minimum waiting time in each network. The minimum communication/access delay is achieved by minimizing the distance between the base stations and edge servers, and the workload balancing helps to reduce the waiting time.

Edge server placement and resource allocation are NP-hard combinatorial optimization problems, and by considering both of them simultaneously, the problem becomes more challenging [28]. Algorithms such as mixed-integer programming (MIP),

heuristics, and clustering algorithms are used to solve this problem. Combinatorial optimization consists of searching and finding the optimal option among a limited set of discrete possible solutions. Recently, reinforcement learning (RL) [45] has shown promising results in this domain where the combinatorial problems are defined as a sequential decision problem. Examples of these problems include traveling salesman to find the shortest path, mixed-integer linear programming, graph coloring problem, and Knapsack problem [27]. Moreover, there are a few studies that have considered the use of RL for placement optimization [30, 31, 33, 46].

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 [27, 14]. Considering the gap in the literature review and Q-learning challenges for real-world problems, providing an efficient RL framework to model the edge servers placement and base stations allocation is our focus in this thesis. We are considering two views for MEC design. The first one is aiming to minimize the cost based on given constraints, including maximum accepted delay for the network and maximum workload capacity for edge servers. Due to the nature of the MEC, which is delay reduction, minimizing the network delay is a vital factor that should be considered as well as minimizing the number of edge servers for having an optimal design. Another view is when a fixed number of edge servers based on the budget is determined. Therefore, the optimal placements should be found to have a high QoS, which can be achieved by creating K-fair-balanced clusters.

1.3 Objectives and Contributions

The main objective of this thesis is to provide efficient RL frameworks for these combinatorial optimization problems to have an optimized MEC design. The output can be seen as different clusters of edge servers and their connected base stations.

In the first model, explained in Chapter 3, the aim is to minimize the cost of the MEC architecture under specified constraints. The considered objectives are:

1. Minimizing the number of edge servers as adding each edge server increases the deployment cost.

2. Reducing the network delay as the main objective in MEC, which increases the QoS.

For the second model, also described in Chapter 3, the following objectives are aimed to be achieved:

1. Minimizing the delay for task computation having a real-time response, which is the main object of the MEC.

2. Minimizing the difference of clusters delay to have the fair allocation and the same QoS in all areas of the network.

3. Minimizing the difference of edge servers workloads, which is equal to the total workload of the connected base stations, by assuming identical edge servers in a homogeneous network.

Using RL for placement is highly challenging and the number and distribution of the base stations, the constraints, and the properties of edge servers are the factors that increase the difficulty [14]. 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 [14, 13]. Furthermore, in our problem, there is not a fixed situation as a goal for the agent, thus the reward value cannot be backtracked. However, there are different research efforts that have considered the RL challenges, but there are a few studies that have considered different challenges in one algorithm [13]. Hence, the contributions of our research are as follows:

- Providing a novel RL framework for the joint problem of edge server placement and workload allocation.

- Combining Q-learning with K-means and top-K to improve the results.

- Defining the state space, action space, and penalty function for multi-objective problems without a fixed end-goal.

- Adopting the base reinforcement learning algorithm for a real-world problem for a large number of base stations.

1.4 Thesis Outline

The remaining parts are organized as following. In Chapter 2, after having an overview of RL and Q-learning, we review the most relevant papers related to our work

in the MEC design and the RL application for combinatorial optimization problems. In Chapter 3, our two mathematical models are proposed. Then, our RL frameworks for edge server placement and computation offloading are explained in detail. The issues and limitations of Q-learning implementation and the offered solutions are clarified. Chapter 4 gives extensive experiments, performance evaluations, and discussions after describing the real-world dataset. Moreover, the other methods which are used for performance comparison are explained. Finally, in Chapter 5, a summary of the research and some directions for future work conclude the thesis.

Chapter 2

Background and Literature Review

This chapter consists of four main sections. First, since Q-learning is adopted to solve our placement optimization problem as a sequential decision making problem, an overview of reinforcement learning and then Q-learning is given in Section 2.1. Next, Section 2.2 provides a review of proposals in the MEC design with respect to the edge server placement and computation offloading. This section starts by discussing some studies that have only considered task offloading, then the solutions that have applied RL in this area. In Section 2.3, different approaches using RL for the placement optimization are discussed. Finally, in Section 2.4, a discussion of the reviewed work highlighting some gaps and limitations is given.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning with a learner called agent. The agent makes decisions by interacting with its surrounding, which is called the environment. The environment sends the reward or penalty signal and the new state to the agent after taking each action (Fig. 2.1). Through that, 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, action space, reward, state transition probability, and discount factor should be defined through the MDP tuple $\langle S, A, R, \gamma, P \rangle$.

- Set of states S. The situation at each time step, it can be discrete or continuous.

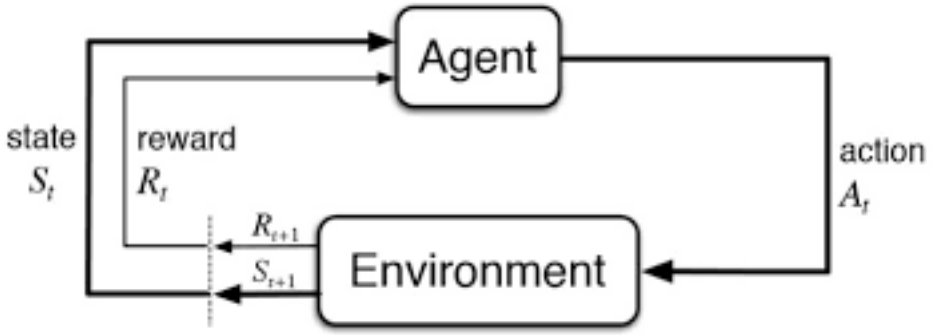


Figure 2.1: The reinforcement learning framework.

- Set of actions A. The choices of the agent in each state that could have an impact on the immediate reward, next state, and future rewards.
- Reward R. A numerical value that is received from the environment to motivate the agent for taking optimal actions including immediate reward and delayed reward.
- Discount factor γ. A value in $[0, 1]$ to define the importance of delayed reward versus immediate reward that helps avoid infinitive cumulative reward.
- State transition probabilities P. The probability of transitioning from one state to another state.

Recently, RL has been successfully used to solve real and complex network problems such as resource scaling of containers in fog clusters [39] and scheduling federated learning tasks on IoT devices [38]. In RL, the agent's objective is to maximize the cumulative reward, which shows the effect of all actions in a sequence. It means that the agent needs to find the optimal policy that maximizes the expected rewards, where a policy is a function π that specifies the action $\pi(s)$ that the agent will choose when in state s. Besides exploiting the learned policy, the agent also needs explorations because there is a possibility of finding actions with more rewards. The exploration-exploitation dilemma is still an open question in RL, however, different methods like epsilon-greedy, upper-confidence bound, and optimistic initial values are widely used.

The RL methods are divided into two categories: model-based and model-free. In the model-based methods, the environment and the transition probabilities should be known, or at least be easy to learn; but in model-free methods, the agent's information

is gathered through trial-and-error. On the other hand, the RL algorithms can be on-policy or off-policy. In on-policy algorithms, the focus is to approximate the policy, but in the latter one, the value function is approximated. The value function measures the performance of the policy function. The environment can be episodic, and after a specified number of iterations, the environment restarts. Moreover, it can be continuous where considering the γ value is necessary to avoid infinitive rewards.

Q-Learning

Q-learning [50] is one of the first major RL algorithms, which is a model-free, off-policy algorithm defined by the following Bellman equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{1}$$

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. R_{t+1} is the value of the immediate reward after taking each action.

Q-learning is known as a temporal difference (TD) learning method. It updates the policy by bootstrapping from the current estimate of Q-value, $Q(S_t, A_t)$. It solves the Bellman equation by sampling from the environment and iteratively applying the Bellman optimality equation. In other words, it updates the Q-value over iterations in order to minimize the TD or Bellman error, $[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$.

Q-learning is an off-policy algorithm because the behavioral policy (π_b) is different from the target policy (π^*). The behavioral policy represents the way that the agent acts. The target policy, as its name implies, is the optimal one that needs to be discovered. For π_b, the agent employs the exploration-exploitation dilemma, while for the π^*, it acts greedily. It should be mentioned that by using the Q-learning Eq. 1, there is no need for transition probability because the next action is the one that maximizes the next state's Q-value. To record the calculated Q-values from Eq. 1, a matrix, called Q-table, is created. It consists of states in rows and actions in columns. The agent uses this look-up table for selecting the optimal actions to reach $Q^*(S, A)$.

2.2 Edge Server Placement and Computation Offloading in MEC

Mobile Edge Computing or Multi-access Edge Computing (MEC) is a new computing paradigm where servers are located close to the end devices, within the mobile access network [49]. These servers are located at the edge of the Internet to process a large amount of data in a distributed manner, and thanks to this proximity, latency decreases while responsiveness increases [43, 36, 2]. In fact, MEC plays a vital role in the development of the 5G network. Additionally, it is a necessary platform in the IoT era since IoT devices generate a tremendous amount of data that needs to be processed locally [17].

Base stations equipped with server or cloudlet are known as the edge servers, and the mobile users, IoT devices, or sensors offload their tasks to these servers [36]. Edge servers, as a critical part of MEC, have received attention among researchers in recent years. As mentioned earlier, we classify the studies about edge servers into two categories. The first category only considers the computation offloading, while the second category studies the service placement and computation offloading as a joint problem.

2.2.1 Computation Offloading

Wei *et al.* [51] state that papers in the task offloading focus on two main objectives: reducing the execution delay and decreasing the energy consumption in mobile devices. Furthermore, determining which tasks should be offloaded and where are the two major decisions needed to be made [51]. The execution delay comprises the time for sending out the data to the servers, processing time, and time of receiving the processed data [26]. For energy consumption, the computation offloading increases the battery life of mobile devices because the tasks are not processed locally. On the other hand, there is energy consumption in transmitting the data and receiving the results [26]. In [20], the goal is to reduce the average energy consumption of mobile devices by considering the delay constraints. The authors modeled the problem as a constrained MDP, and they examined two strategies. The first one is using online learning, and the network adapts to the changes in mobile devices dynamically. The second strategy is using the partial network's knowledge after specified time intervals.

Their results show that the pre-calculated offline approach, despite having imperfect information, has better performance.

A power-constraint problem to minimize the delay for a single user is studied in [25]. The mobile device decides whether to process the task locally or offload it to the server based on a stochastic policy. The experiments demonstrate that the proposed MDP modeling outperforms when compared to the situations when all tasks are computed locally, or all of them are sent to the server, or when the mobile device makes a greedy decision. The authors in [34] have offered a two-phased approach to reduce the complexity of the problem. In the first step, the mobile device searches its Wi-Fi region to discover available cloudlets, and in the second step, based on the task's requirements, the cloudlet is selected. Some studies, such as [32], have considered the multi-cloudlet environment where reducing the power consumption and latency are the main factors for choosing the optimal cloudlet.

Some studies have investigated the trade-off between the two mentioned objectives: delay and energy consumption. For example, in [9], the investigators have considered one user with independent tasks, and like [25], there is a binary offloading decision. There is one more assumption that if the task is not computed locally, there is another binary decision, which can be MEC or remote cloud. The goal is minimizing the cost, which consists of energy computation and delay, by using a heuristic algorithm. The near-optimal performance was attainable after a few randomization iterations.

2.2.2 Reinforcement Learning for Computation Offloading

In [10], 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 [42] also used Q-learning with only 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. The authors showed that their approach could reduce execution time and energy consumption. Moreover, the decisions are made in near real-time.

For the joint optimization of task offloading and bandwidth allocation, in [18,

54], a deep-Q network is used to learn the offloading policy where the objective is minimizing both the latency and energy consumption. In [54], a neural network, based on receiving data from the environment, is trained and then modeled to predict the action in each state. For the bandwidth allocation, the authors have separated each task into two parts: a part that is computed locally and a part that is sent to the server. Then, the server is selected, and their algorithm finds the optimal percentage of the task that should be offloaded.

All the previous investigations are based on Q-learning, which is an off-policy approach. An on-policy reinforcement learning algorithm called SARSA (State-Action-Reward-State-Action) is used in [2] to reduce the energy consumption and execution delay. The state at each time step consists of the uploading and downloading bandwidth. There are three possible actions: selecting the nearest edge server, the adjacent edge server, or the remote cloud.

2.2.3 Edge Server Placement and Task Offloading

To find the minimum number of cloudlets while covering the largest number of mobile users, Peng *et al.* [35] used an improved affinity propagation algorithm. In their clustering, they considered the 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. To define the optimal number of cloudlets to have a trade-off between the QoS and the cost for the service provider, they used affinity propagation based on the preference value of nodes. Finally, a directed acyclic graph shows the new place of cloudlets, and the number of new cloudlets would be adjusted. The number of covered users and the difference in the number of mobile users in each cluster were used, respectively, to evaluate the coverage and load balancing. The authors generated a dataset for different numbers of mobile users, and their algorithm outperformed K-means and mini-batch K-means.

The authors in [19] have modeled the problem of cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks as a queuing network to reduce the average waiting time while maintaining a balanced workload. They used the heaviest-AP first algorithm with the density-based clustering by the assumption that the number of cloudlets is given. Then, K-median is used to find the cloudlet

places with the minimum delay. Their simulation environment is created based on the dataset of the wireless network in Hong Kong for 18 distinct areas (nodes). In their modeling, each cloudlet is known as a set of homogeneous servers connected to the cloud. Part of the tasks can be offloaded to the remote cloud when one of the cloudlets is overloaded. They also have assumed the cloud has adequate computation power for all the offloaded tasks, and there is no queue time.

The focus of the two previous studies was on the cloudlet placement. Wang *et al.* [49] have proposed the first study for the edge server placement. The authors 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 dataset, 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 [15], for the same dataset, they have used the MIP and K-means combination for 20 to 200 base stations. In one of their investigations, they have increased the number of edge servers for the fixed number of base stations. In the case of 5 edge servers for 200 base stations, the top-K outperforms their model, but for the other experiments, they have improved the results. Also, for the same dataset, reducing the total energy consumption, by assuming the linear relation between CPU utilization and energy consumption, is studied [24]. Particle swarm optimization is adopted for up to 1300 base stations. The results were compared with random and top-k algorithms for the edge servers with different coverage areas. A particle swarm optimization algorithm is adopted, and the results were compared with random and top-k algorithms for up to 1300 base stations by considering different coverage area for edge servers.

Cao *et al.* [6] emphasize that in addition to minimizing the response time, the heterogeneity and response time fairness need to be considered. In their two-stage method, in the offline stage, integer linear programming is used to find an optimal placement strategy for the edge servers. Then, in the online one, a game theory-based method is adopted to consider the workload fluctuation of the base stations.

2.3 Reinforcement Learning for the Placement Optimization

The edge server placement and resource allocation is a constrained combinatorial optimization problem. The combinatorial optimization problem (COP) is searching and finding the optimal option among a limited set of discrete possible solutions. The exact methods and heuristics can be used to find it. Moreover, in terms of machine learning methods, supervised and reinforcement learning have been used [12]. Different types of combinatorial problems are: Traveling salesman, Mixed-Integer Linear Program (MILP), generation, and enumeration [27].

This survey [27] shows that the combination of RL and COP can be a promising method for solving combinatorial problems. However, one of the main difficulties in CO is the state-space explosion [7].

Despite that, there are a few studies that have applied RL for the placement. Mirhoseini *et al.* [31] 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. Addanki *et al.* [46] have studied the same problem, but using the graph embedding method has helped to generalize and improve the algorithm to be applicable for different neural networks rather than only a specific one. The algorithm's output is policy instead of places for operations. This learned policy is transferable for the unseen neural networks of the same family. In both [31, 46], before implementing the RL, they have grouped the same operations and forced the algorithm to place them in the same device, which reduces the placement actions.

Mirhoseini *et al.* [30] proposed using RL for chip placement. The performance of their algorithm improves over time by having more experience. Same as [46] their algorithm can be used for unseen blocks, even with the bigger size. Unlike the [46], in this study, the placement policy completes 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, they 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 dataset to train it in a supervised manner. They have noticeably improved the placement time while outperforming other methods.

2.4 Discussion

In this chapter, we provided a review of the relevant studies to our work for MEC design, considering the computation offloading, placement optimization, and RL application for the placement optimization. The demand for real-time processing for computation-intensive tasks has led to the emergence of the MEC paradigm. In the reviewed studies for computation offloading, the action-space is limited. Most of these solutions have not analyzed the problem of which server should be selected if a task is offloaded to the edge. In other words, their modeling does not include the detail of the task allocation when the edge is elected as a destination for offloading.

To find the optimal placement and resource allocation in MEC, various approaches are investigated, including mainly: clustering [35], top-K [19], MIP [49], combination of MIP and K-means [15], heuristics [24], ILP and game theory [6]. The proposed models are implemented on different numbers of base stations, going from 18 to 3000, on a real-world or generated datasets.

As mentioned earlier, to the best of our knowledge, in the current literature, applying RL for the edge server placement has not been investigated yet. Furthermore, most proposals on applying RL to other placement optimization have examined the problems which need a real-time response. In our problem, the edge server placement optimization is different from the studies in Section 2.3 because we have to choose fixed places that do not vary over time. This is like the famous maze problem that the agent, after enough iterations, needs to find the optimal policy to reach the goal. However, unlike maze, there is no fixed points 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. Therefore, in this thesis, a novel Q-learning-based approach is examined and applied to design MEC with regard to optimized placement of edge servers while considering the deployment and delay cost either as a variable to be minimized or a constraint to find optimal places for K-edge servers.

Chapter 3

Edge Server Placement Modeling and RL-Frameworks

This chapter starts by describing the problem and demonstrating the MEC model in Section 3.1. Next, in Section 3.2, the mathematical formulations of two different points of view in designing MEC concerning the optimal edge servers placement are provided. Then, our proposed RL-frameworks for these sequential decision making problems and the steps to solve the encountered issues are explained in detail in Section 3.3. Finally, Section 3.4 presents a summary and comparison of our solutions.

3.1 System Model and Problem Definition

MEC is composed of users/mobile devices, base stations, and servers (Fig. 3.1). There are many users/mobile devices distributed through the city which are connected to the base stations with limited coverage area to process the data. Edge servers with more computation power are added to the network to process computation-intensive tasks. Base stations offload the received tasks from the users to the edge servers instead of sending them to a remote centralized cloud. Thus, equipping the network with servers close to the end-users leads to the principal purpose of the MEC paradigm, namely the real-time response.

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 network can be shown by an undirected graph $G = (V, E)$, where $V = BS \bigcup ES$ and

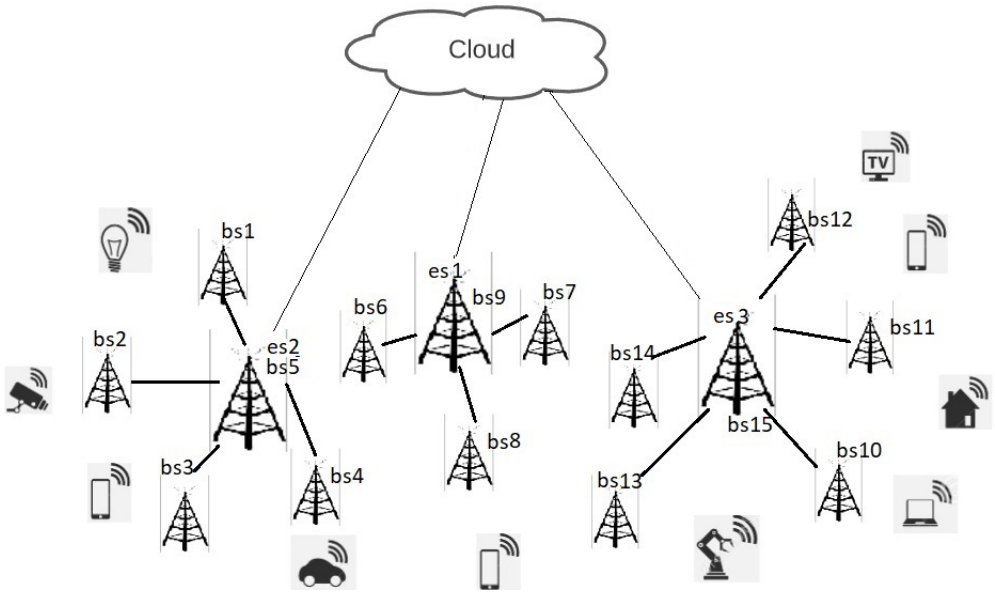


Figure 3.1: Edge-server placement in MEC.

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 [49, 35].

In Fig. 3.1, 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. The connections among base stations or the edge servers are not included in our modeling.

Adding the edge servers increases the computation power and decreases the latency, but then it subjects the network design to some restrictions. First, edge servers are limited in number because of the deployment cost and their energy consumption. Second, each of these servers covers a limited number of base stations based on their workload capacity. These limitations make the design of the MEC architecture a complex challenge.

In order to have an optimal network design, two models are discussed in this thesis. In both models, the following assumptions are considered:

- Each edge server is co-located with one of the existing base stations.
- 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.

- The distance between the base station and its edge server represents the delay.

There are two main objectives for both models. The first objective is about finding the optimal placement for the edge servers. The second objective is finding the optimal assignment of the base stations to the edge servers by creating the optimal clusters. These two problems are known to be NP-hard [49]. This thesis introduces a reinforcement learning-based approach to reach the aforementioned objectives thanks to its promising results in combinatorial optimization problems [27].

Our study does not include the connections between users and base stations as well as edge servers and the cloud. Moreover, in this thesis, the MEC design refers to finding the optimal edge server placement and base station allocation/workload offloading by using the historical data from a dataset. This data, explained in Chapter 4, is gathered from users' connections to the base stations over days. The key variables used in our formulations and modeling are shown in Table 3.1.

3.2 Edge Server Placement Formulations

3.2.1 First Model: Minimizing the Cost of MEC

In our first model, finding the optimal MEC design with minimum cost is investigated. Deployment of new edge servers increases the cost of the network. Moreover, since the principal purpose of MEC and 5G networks is to bring the computation power near the end devices in order to have a real-time response, striving to have a minimum delay is one of the most vital objectives that should be considered. In fact, as argued in [2], the long execution time is one of the main concerns in the design of MEC.

Therefore, minimizing the network's cost relies on two parts: minimizing the number of edge servers K (Eq. 2), and minimizing the network delay D_N (Eq. 3). This problem is modeled as follows:

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
$\|n_j\|$	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
Λ	set of all edge servers' workloads in the network
Λ_j	workload of edge server j in the network
λ	set of all base stations' workloads in the network
λ_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
DoF_i	number of connected nodes, degree of freedom
$ne_{1_{bs_i}}$	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
z	binary variable if selected base station is a new edge server $z = 1$
Pr_i	priority of bs_i

Table 3.1: Notations.

$$O_1 : \min(K) \tag{2}$$

$$O_2 : \min(D_N) \tag{3}$$

subject to the following constraints:

$$\Lambda_j \leq \Lambda_{max} \quad (\forall j, 1 \leq j \leq K) \tag{4}$$

$$\sum_{j=1}^{K} (x_{ij} d_{ij}) \leq D_{TH} \quad (\forall i, 1 \leq i \leq n) \tag{5}$$

$$\sum_{j=1}^{K} x_{ij} = 1 \quad (\forall i, 1 \leq i \leq n) \tag{6}$$

$$x_{ij} \in \{0, 1\} \quad (\forall i, 1 \leq i \leq n \ and \ \forall j, 1 \leq j \leq K) \tag{7}$$

To find the optimal placement and the optimal clusters, some constraints are applied to the model that needs to be satisfied. First, each edge server has a limited computational power, thus it can handle a limited workload. An upper bound, Λ_{max}, is considered for the edge servers as the processing capability (Eq. 4). Second, a maximum acceptable delay is considered in the constraints (Eq. 5). As the distance represents the delay, D_{TH} is used to represent the distance threshold. It is worth to note that, although an acceptable constraint for the latency is considered (Eq. 5), as mentioned above in Eq. 3, still our goal is always to further minimize the delay. Finally, each base station is connected to just one edge server while all the base stations are covered (Eq. 6) and x_{ij} is a binary variable (Eq. 7) and $x_{ij} = 1$ if base station i is connected to the edge server j; otherwise, $x_{ij} = 0$ for all $1 \leq i \leq n$ and $1 \leq j \leq K$.

3.2.2 Second Model: K-edge Server Placement

In the second model, the placement of K-servers, and base station allocation are studied. It is assumed that the service provider, based on the budget limitation, gives the number of edge servers, K. Moreover, latency reduction is one of the main objectives of MEC, and the distance between the base station and the edge server shows the access/communication delay. Thus, in this model, the best dominant area

for each edge server should be defined to have the minimum latency. Finally, because finding the optimal base station allocation should be investigated, the ultimate results would be K clusters of the edge servers and their connected base stations.

Similar to studies in [15, 24, 49], it is assumed that all the edge servers have the same computation power (a homogeneous network). Therefore, the clusters should have a balanced workload which prevents the overloaded, under-loaded, or idle edge servers. Moreover, minimizing the difference between the clusters' average delay is necessary to have fair allocation [6] and the same QoS in all areas. The model of this multi-objective problem is as follows.

$$O1 : \min \frac{\sum_{j=1}^{K}\sum_{i=1}^{n}(d_{ij} \times x_{ij})}{n} = \min \frac{\sum_{j=1}^{K}\overline{D}_j}{K} \tag{8}$$

$$O2 : \min\{\max(\Lambda) - \min(\Lambda)\} \tag{9}$$

$$O3 : \min\{\max(\overline{D}) - \min(\overline{D})\} \tag{10}$$

subject to the following constraints:

$$\sum_{j=1}^{K} x_{ij} = 1 \quad (\forall i, 1 \leq i \leq n) \tag{11}$$

$$x_{ij} \in \{0, 1\} \quad (\forall i, 1 \leq i \leq n \ and \ \forall j, 1 \leq j \leq K) \tag{12}$$

The average distance between edge servers and their covered base stations is considered as latency, and Eq. 8 is defined to have minimum communication delay. By considering a homogeneous network Eq. 9 is applied to have balanced clusters, where Λ is a set of edge servers' workloads $\Lambda = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_K\}$. The workload of each edge server is equal to the total workload of all the connected base stations. For example, for cluster K:

$$\Lambda_K = \sum_{i=1}^{n}(\lambda_i \times x_{iK}) \tag{13}$$

λ_i is the workload of base station i.

$\overline{D}$ in Eq. 10 is a set of average distances in each cluster, $\overline{D} = \{\overline{D}_1, \overline{D}_2, \ldots, \overline{D}_K\}$. Each element of this set represents the average distance of the base stations from the edge server in each of the K clusters.

$$\overline{D}_j = \frac{\sum_{i=1}^{n}(d_{ij} \times x_{ij})}{|n_j|} \tag{14}$$

and

$$x_{ij} = \begin{cases} 1, & \text{if } bs_i \text{ is in the cluster j} \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

where $1 \leq j \leq K$. The distance of base station i from edge server j is d_{ij} and $|n_j|$ is the number of base stations in cluster j. Eq. 10 by decreasing the difference in the cluster's delay creates a fair allocation in the network. Similar to the previous model, the constraint (Eq. 11) demonstrates that each base station is only connected to one edge server, while all the base stations are covered. And similar to our previous model x_{ij} is a binary variable (Eq. 12) for all $1 \leq i \leq n$ and $1 \leq j \leq K$.

3.3 Modeling the Placement as an RL Problem

The joint problem of finding the optimal placement and workload offloading is modeled as an RL problem. As RL is based on MDP, defining the state space, action space, and the reward function are the fundamental steps.

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.

3.3.1 RL Framework for Minimizing the Cost of MEC

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 coordination: latitude and longitude. The initial state and the location are formalized as follows:

$$Initial\_State\_set = [\ L_{bs_1},\ L_{bs_2}, \ldots,\ L_{bs_n}\ ]$$
$$L_{bs_i} = [\ lat_{bs_i},\ lon_{bs_i}\ ]$$

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. 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:

$$A = \{a_{bs_1}, a_{bs_2}, \ldots, a_{bs_n}\}$$
$$a_{bs_i} = [ne_{1_{bs_i}}, ne_{2_{bs_i}}, \ldots, ne_{N_{bs_i}}];$$

where a_{bs_i} shows the action space of base station i which is limited to its neighbors and $ne_{1_{bs_i}}$ 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. Variable action space is a challenging situation in deep reinforcement learning [5, 8]. 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 Q-table.

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:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \times [\mathbf{P}_{t+1} + \gamma \mathbf{min}_a Q(S_{t+1}, a) - Q(S_t, A_t)] \qquad (16)$$

P represents the penalty, and the agent's policy based on $\min_a Q(S_{t+1}, a)$ is minimizing the penalty, instead of maximizing the reward. We considered $\alpha = 0.4$ and $\gamma = 0.9$

for our algorithm.

Goldie and Mirhoseini [14] assert that penalty/reward calculation for the placement problems when the action space is not completed (partial placement) is very difficult. Further, for multi-objective problems, it is usually hard to determine a single penalty value. And to evaluate the policy, it is crucial to consider the performance of the algorithm for each objective separately [13].

Distance (Eq. 26), as the first factor, was used in calculating the penalty because it represents 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 (Eq. 2). 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 in Chapter 2, in the maze problem, there is a point when reached, it brings a high reward. Then the agent can learn the optimal policy by backtracking the reward value. However, in our problem, there is no such a point. 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) [29]. 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.

To address the two aforementioned issues: lack of end goal and 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:

$$Modified\_State_t = [L_{bs_t}, [a_{s_1}, a_{s_2}, \ldots, a_{s_{t-1}}]]$$

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.

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:

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

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

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 deep Q-network. Hence, our priority is using the Q-table.

Though our modeling solves the mentioned difficulties, namely: large and variable action space, penalty calculation, CAP, and lack of exact end goal, when using the

Q-table, we face a large state-space. In the Q-table, the states are the rows, and the actions are columns. In our state definition, there is a fixed part that is the base station location, which is the same in all the episodes. The other components vary based on different actions. In our Q-table, only the fixed part is considered as the state for being fitted in the look-up table (Algorithm 1. line 5). Finally, the goal is to have the best action for each base station, and this method helps us have the average Q-value for each state in different situations. It should be noted that in different episodes, the agent uses the other parts to calculate the penalty and for taking the 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 [45].

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:

$$Pr_i = (\lambda_i + DoF_i)/average\_distance_i;\ \ (\forall i, 1 \leq i \leq n) \tag{17}$$

where λ_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. $average\_distance_i$ 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:

$$P = distance(L_{bs_i}, L_{a_{bs_i}}) + fixed\_value \times z + 1/Pr_{a_{bs_i}} \tag{18}$$

where the distance is calculated based on Eq. 26, which will be explained in the next chapter. The input of distance function 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 Eq. 18. 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.

So far, the fundamental parts for creating the efficient classic Q-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:

$$\epsilon = 9/(T + 100) \tag{19}$$

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.

Algorithm 1 represents our proposed Q-learning framework for minimizing the cost (QMC). ResetEnvironment() refers to the process of updating the edge servers list in each episode ($ES = [\,]$), updating the workload dictionary of the selected edge servers ($wl_{es} = \{\,\}$) 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.

Algorithm 1: QMC Framework

```
Initialize Q(s, a) to 1000 except adjacent nodes for each state to 0, and
  Q(terminal, .) = 0
for episode ← 1 to MAX do
    ResetEnvironment()
    for i ← 1 to n do
        s_i ← L_{bs_i}
        if s_i in ES then
            a_{bs_i} ← bs_i, obtain P, s_{i+1} ← L_{bs_{i+1}}, update Q, update ES, update
              wl_ES
        else
            PA ← GetPossibleActions()
            Q_PA ← GetQvaluesofPA()
            if |PA| > 0 then
                Choose a_{bs_i} for s_i using ε-greedy policy from Q_PA
                take action a_{bs_i}, obtain P, s_{i+1} ← L_{bs_{i+1}}, update Q,
                  UpdateEnvironment()
                /* UpdateEnvironment(): update ES, update wl_ES and update
                   FA                                                    */
            else
                Nei ← GetNeighbors()
                Q_Nei ← 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} ← L_{bs_{i+1}}, update Q,
                  UpdateEnvironment()
                /* UpdateEnvironment(): update ES, update wl_ES and update
                   FA                                                    */
```

In summary, basic Q-learning is adopted to solve the placement and allocation problem. Then, by considering some rules to select the actions and adding the history of actions to the state space, the search tree became smaller. Next, the possible-action vector that is built up through each episode is added as an additional step to our

algorithm. These steps, besides an effective definition of the penalty function, made the Q-learning applicable to our large scale problem.

3.3.2 RL Framework for K-edge Server Placement

For this problem, the initial state space was defined as a set of all base stations' locations consisting of latitude and longitude as follows:

$$Initial\_State\_set = [L_{bs_1}, L_{bs_2}, \ldots, L_{bs_n}]$$

This state definition cannot help the algorithm converge to the optimal value. The reason is that taking a particular action in a specific state does not have the same result in all the episodes. As the action in each state is dependent on the others and there are many possible combinations, the agent is not able to find the optimal action.

The state definition needs to be modified to tackle the above issue. A list which shows the action of each base station is added to the state definition (Algorithm 2, line 5). Thus, the state apace at each time step is:

$$State_t = [L_{bs_t}, [a_{bs_1}, a_{bs_2}, \ldots, a_{bs_{t-1}}, \ldots, a_{bs_n}]]$$

where $1 \leq t \leq n$, and the actions for the base stations before time step t are the updated actions while the others are the pre-assumed allocations. In our work, an initial list of actions is created by assuming that each agent has selected itself as the edge server (Algorithm 2, line 3) as follows:

$$Action_0 = [bs_1, bs_2, \ldots, bs_n]$$

However, this is far from the optimal action. It will result in a high penalty, but along the path after taking action for each base station, the better decision replaces the initial allocation. Over iterations, the optimal actions will be discovered by the agent.

All the base stations including the current base station where the agent is located could be considered as the action space. Each of these base stations have the

chance to be selected as the edge server. The issue here is facing a large action space specially when the number of base stations increases. This huge action space decreases the efficiency of RL-agent performance because the agent has to explore more, which increases the computation, or it has to ignore some of the actions and rely on exploitation.

To reduce the action space in our problem, the nodes which are not part of the optimal solution are eliminated. Although there is not a distance threshold as a constraint, for a large number of base stations, just the neighbors are considered. For example, for base station 1, there are some base stations close to it, for instance, within 15 *km*. These nodes are better actions for being taken compared to the nodes outside of this vicinity, as the main objective is reducing the latency. Determining this value is dependent on the base station's closeness. This value should be increased when the base stations are not located close to each other in order to have feasible actions. Otherwise, the action space becomes empty for some base stations. Same as the previous model, we face a variable action space for each state.

The penalty function should be defined in a way to lead the agent to accomplish the objectives because the agent takes actions that minimize its cumulative penalty. As the agent should achieve three different objectives, the penalty function is composed of different parts.

For the delay, the second part of state space is used. The distance of each node to its action is calculated based on the geographical location of the nodes (Eq. 26). For example for 4 nodes, if the second part of the state is $[bs_1, bs_1, bs_3, bs_3]$ and the order of the nodes that the agent is passing through is $[bs_1, bs_2, bs_3, bs_4]$, the distance penalty is:

P_{Dis} = [scaled_distance(L_{bs_1}, L_{bs_1}) + scaled_distance(L_{bs_2}, L_{bs_1})
+ scaled_distance(L_{bs_3}, L_{bs_3}) + scaled_distance(L_{bs_4}, L_{bs_3})] / 4

Here, the agent tends to select the K first base stations because the delay is zero. Consequently, the actions are dependent on the node orders. To address this issue, a value called *average-distance*, for each node, is calculated. This is the average distance of each node to its neighbors by using the scaled distances. The neighbors are considered the N nearest nodes. As it is considered $K = n/10$ like the work

in [49], N is 10 based on the following Equation:

$$N = (\text{number of base stations})/K = n/K \tag{20}$$

Then, Eq. 9 for having fairness, and Eq. 10 for having a balanced workload are added to the penalty function.

Two kinds of action spaces are not accepted: the one with more or fewer than K edge servers and the other one when a base station is both sender and receiver in an episode. In a path in each episode, a base station is a sender if the agent chooses another base station for offloading. Then, the selected base station is the edge server and is the receiver. The agent will be penalized (P_{sr}) if there is a node with the mentioned condition.

Therefor, the final penalty function is:

$$\begin{aligned} P_{a_{bs_t}} = (average\_distance)_{a_{bs_t}} + \{\max(\Lambda) - \min(\Lambda)\} + \{\max(\overline{D}) - \min(\overline{D})\} \\ + (\sum_{i=1}^{n}((distance(L_{bs_i}, L_{a_{bs_i}})))))/n + w \times (P_{sr}) \end{aligned} \tag{21}$$

where w is a binary variable and is equal to 1 if a node is both sender and receiver. It should be noted that in calculating the penalty the scaled value of the distance and workload is used.

For ensuring that there is only K servers, in the beginning, the agent is allowed to take different actions (Algorithm 2, from line 11 to 14), but after having K edge servers, the action space will become limited to that K nodes (Algorithm 2, from line 8 to 10).

In our algorithm, a Q-dictionary is created rather than the Q-table because the states are not known in advance. In our Q-dictionary, the string of state is the "key" at each time step. A list filled with high values (1000) is the "value" for each new state, except the neighbors' entries that are initialized to zero (Algorithm 2, lines 6-7 and lines 15-16). This optimistic initialization besides the ϵ-greedy policy motivates the agent to explore possible actions over iterations [45].

Based on Eq. 16, the Q-dictionary values update over iterations. The $\gamma = 0.3$ and $\alpha = 0.5$, for the first experiment for 20 nodes. As mentioned, a higher value for γ represents the importance of actions in the future states. By considering our penalty function and the action selection limitations, the immediate and recent penalties are more important. Besides, if the nodes are far, their actions do not affect each other.

For each node, the action space is limited to its neighbors. Therefore, the neighbors' actions need to be considered, not the action of all the remote nodes. As the number of base stations increases and the network becomes dense, reducing the γ and the threshold distance is necessary. Otherwise, we face the infinity value for the cost.

The following example gives an overview of our Q-learning algorithm (Algorithm 2) for the placement and optimal allocation for K-edge servers (QPAK). The RL-agent starts its path from the first base station, and the state is: $\{[a_1, a_2, \ldots, a_n], 1\}$. As mentioned earlier, in the first state, it is assumed each base station has selected itself as the edge server for offloading, hence the state is: $s_1 = \{[bs_1, bs_2, \ldots, bs_n], 1\}$. Then, the agent will take an action based on the rules and limitations. For example, bs_5 is selected as the destination for the first node. Then, the new state will be $s_2 = \{[bs_5, bs_2, \ldots, bs_n], 2\}$. It shows the agent has moved to the second node (2). Then the penalty based on Eq. 21 will be calculated. The agent receives the numeric value as the result of that action. It continues until passing all the base stations and assigning them to an edge server. After enough iterations for exploring different possibilities based on the epsilon value and optimistic initialization, after converging the cost, the action in each state is determined.

In QPAK, the action space is limited to the node's neighbors. However, as the agent should consider offloading as well as placement, it still causes large action space. Thus, many explorations for different combinations are needed. In this situation, the computation time increases, and more iterations are needed for convergence. Therefore, in addition to QPAK for adopting Q-learning, to further reduce the action space, our second algorithm, named CQPAK[1] (Algorithm 3), which combines the Q-learning with Top-K and K-means is offered. In this approach, the action space is limited to the selected nodes as the edge server for those two models (K-means and Top-K). However, it might reduce the efficiency by not exploring the other nodes, but it might also increase the efficiency by having a smaller action space through receiving the potential nodes for the edge server placement. Top-K offers the nodes with the heaviest workload, and for K-means, in each cluster, the node with the highest priority (Eq. 17) is selected. These nodes are the action space in our CQPAK algorithm, while the other steps of the algorithm are the same as QPAK.

[1]The letter C stands for Combined.

Algorithm 2: QPAK Framework

```
for episode ← 1 to MAX do
    ResetEnvironment()
    Initialize action space in a way that each base station selects itself as edge server
    for i ← 1 to n do
        s_i ← [action space, bs_i]
        if s_i not in Q then
            Initialize Q(s, a) to 1000 except adjacent nodes for each state to 0
        if |ES| = K then
            QK ← GetQofK()
            Choose a_{bs_i} for s_i using ϵ-greedy policy from QK
        else
            QN ← GetQofNeighbors()
            Choose a_{bs_i} for s_i using ϵ-greedy policy from QN
        take action a_{bs_i}, obtain P, observe new action space, s_{i+1} ← [new action space, L_{bs_{i+1}}]
        if s_{i+1} not in Q then
            Initialize Q(s, a) to 1000 except adjacent nodes for each state to 0
        update Q
        action space ← new action space
```

Algorithm 3: CQPAK Framework

```
1  AS = []
2  Implement K-means
3  HPNC ← GetHighPriorityNodesofeachCluster()
4  AS ← AS.append(HPNC)
5  Implement Top-K
6  NHW ← GetNodeswithHighestWorkload()
7  AS ← AS.append(NHW)
8  for episode ← 1 to MAX do
9      ResetEnvironment()
10     Initialize action space in a way that each base station selects a random
         node from AS as edge server
11     Follow line 4 to line 18 of QPAK Algorithm
```

3.4 Summary

In this chapter, two points of view for the design of MEC are proposed. In the first one, the goal is minimizing the cost while maintaining the high QoS. After defining the state space, action space and penalty function in a way to address the faced issues, new steps as rules have been to the classical Q-learning algorithm to make it applicable to our problem.

In the second view, it is assumed that the number of edge servers is given and the goal is having an optimal design to achieve the minimum access delay, having fair and balanced clusters. In addition, an approach based on the top-K and K-means as a first step, before employing the Q-learning is suggested. In the first view, the placement policy is created gradually and over episodes by passing each state, but in the second view our algorithm is used a pre-assumed policy and it is improved over iterations. In these two solutions, the RL difficulties including state space explosion, dealing with large action space, variable action space, lack of static end goal and defining an efficient penalty function for multi-objective problems are discussed and efficient solutions are provided.

Chapter 4

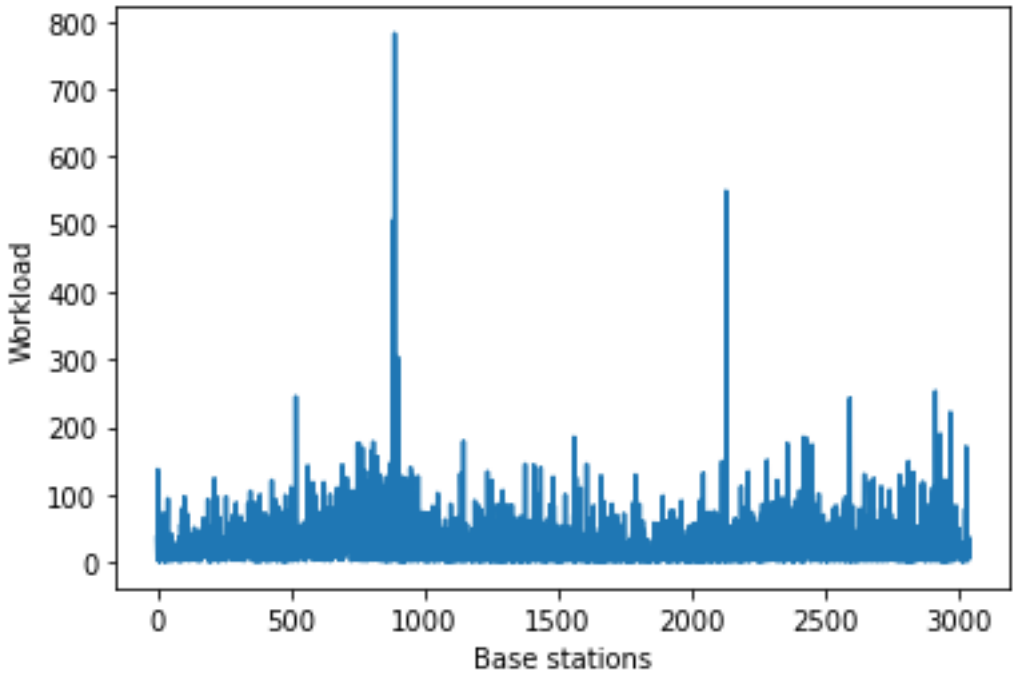
Results and Discussion

This chapter introduces the dataset used in our simulations along with its features in Section 4.1. To validate the proposed RL-frameworks presented in Chapter 3, extensive experiments for different numbers of base stations considering various conditions are discussed in Sections 4.2 and 4.3. The results of implementing our algorithms are compared to other benchmarks to examine the performance of our approaches. Finally, Section 4.4 concludes this chapter.

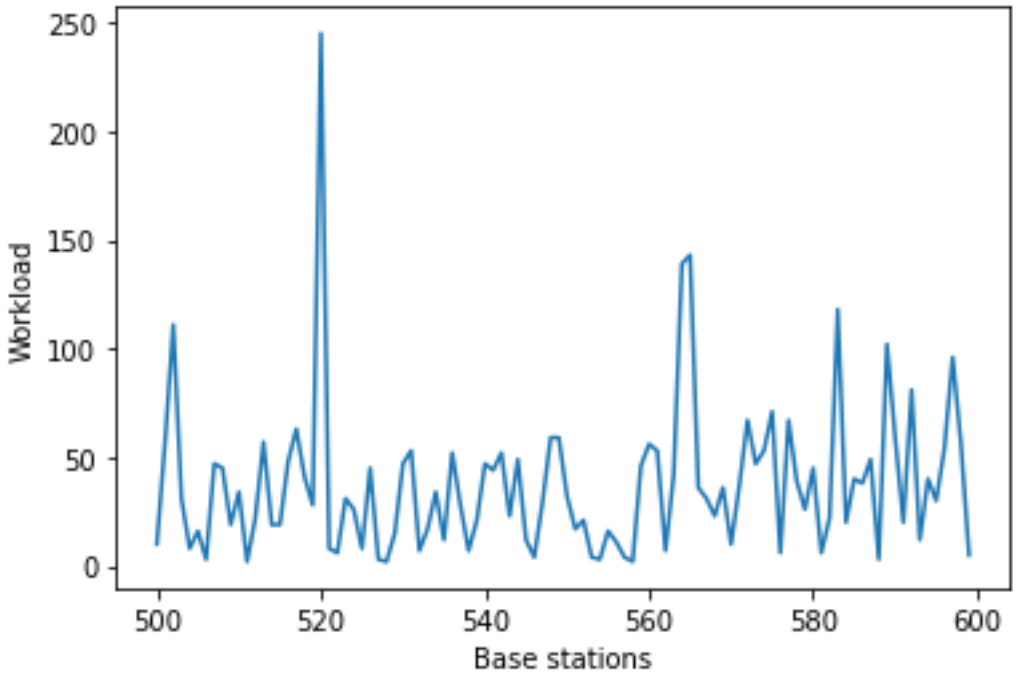
4.1 Dataset

To evaluate the performance of our proposed algorithms, a dataset from Shanghai Telecommunication [49, 48, 15, 47] in China is used. It comprises the information related to 3233 base stations and the connected users over 30 days, in June of 2014. Each row/record indicates a request from a user sent to a particular base station. There are five columns, and the first column shows the date of the requests/tasks, which varies from 1 to 30. The start time and end time of the tasks are, respectively, in the second and third column. The fourth column shows the location of base stations in the latitude/longitude format. Finally, the last column is the user ID of devices that have sent requests to the base stations. After reading the data in Python from an excel file that contains two sheets, the workload of each base station is calculated, which can be quantified in different ways.

In our first model, 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



(a) All the base stations.



(b) 100 base stations (500-600).

Figure 4.1: The workload distribution/the number of tasks of all the base stations and a range of base stations to see more details.

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, $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. 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 [22]. Considering the base stations' workload distribution (Fig. 4.1), some of them with higher workloads were omitted from our dataset.

Since the information of the GPU consumption is not included in the dataset, in our second model, the total duration of the tasks indicates the base stations' workload. However, it should be noted that the performance of the algorithms is not dependent on the method of the workload calculation, and they can be used interchangeably.

As mentioned in Chapter 3, distance represents the delay. The distance matrix is created using the geographical locations in the dataset. For that, the fourth column of the dataset is divided into two separate columns. Instead of Euclidean distance that is the length of a straight line between two points, the following equations are used to have more precise distances:

$$dlon = lon2 - lon1 \tag{22}$$

$$dlat = lat2 - lat1 \tag{23}$$

$$a = (sin(dlat/2))^2 + cos(lat1) \times cos(lat2) \times (sin(dlon/2))^2 \tag{24}$$

$$c = 2 \times atan2(sqrt(a), sqrt(1-a)) \tag{25}$$

$$d = R \times c \tag{26}$$

where lon and lat represent the longitude and latitude of each node in radians, respectively. Eq. 24 is the haversine formula that computes the great-circle distance

between two points in Eq. 25 [44]. This distance is the shortest path between the two points on the surface of the sphere. The $atan2(x, y)$ is the arctangent that gives the radians angle between x and y. And 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 dataset.

4.2 Minimizing the Cost of MEC

One of our primary purposes was to model the joint problem of edge placement and computation offloading as a sequential decision making. After presenting this modeling in the previous chapter, in this section, extensive experiments are conducted to get a view of the model's performance. First, to ensure that the results are not dependent on the location or workload of selected base stations, and second, to analyze the effect of pre-defined constraints, the following actions have been taken:

- Implementing algorithms for different number of base stations, including: 100, 300, 500, 700, 1000, 1400, 1700, 2000, 2400, 2798 base stations.
- Changing the maximum capacity of the edge server to evaluate the performance of the algorithm by considering different workload constraints.
- Changing the distance threshold to evaluate the performance of the algorithm by considering different delays.

After monitoring and validating the model's performance in various situations, the other objective is to see if our algorithm is able to have competitive results in comparison to the other benchmark methods in terms of cost. Therefore, the three popular approaches in the literature, Top-K, Top-DoF, and Random methods are selected for the performance evaluation.

4.2.1 Implementation Results

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* [22].

Based on our proposed algorithm, the agent could find the near-optimal actions for different numbers of base stations, and the convergence is guaranteed after enough iterations (Fig. 4.2). The cost, the y-axis, represents the completed path's penalty in each episode. 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 algorithm worked for large number of base stations (Fig. 4.2f-4.2j) with Q-table without the help of neural networks by keeping a memory of the actions and their results in each episode.

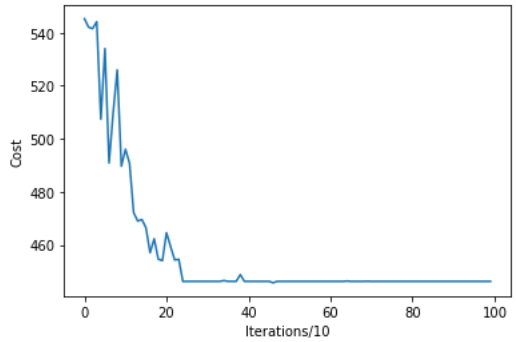
Fig. 4.3 demonstrates the convergence of our algorithm while the distance threshold varies from 3 *km* to 11 *km*. It means the maximum accepted latency varies between 0.01 *ms* and 0.036 *ms*. This experiment is done for 300 base stations, and the maximum workload is considered 150 requests per day. 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. 4.4). Increasing the workload capacity does not necessarily increase the convergence time (comparing Fig. 4.4b and 4.4c), and it is dependent on the node's position and the number of neighbors.

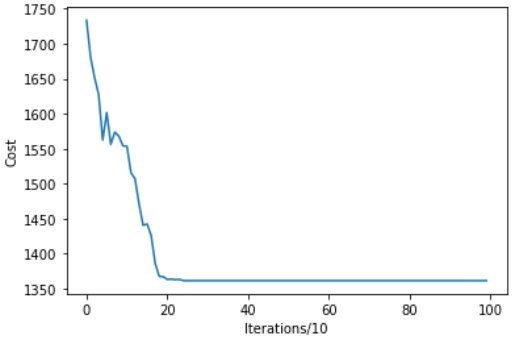
4.2.2 Performance Evaluation

Two performance indicators are used to illustrate the performance of our algorithm, the number of selected edge servers and the average distance, that represent the communication or access delay. It is supposed that the processing time is negligible, and by following the workload condition, there is no queue, and consequently, there is no waiting time. Thus, the access delay represents the network delay.

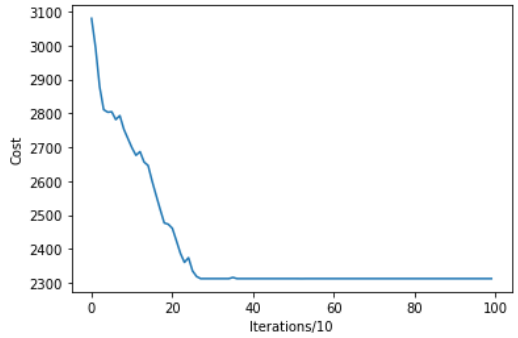
In Top-K, the nodes based on their workloads are sorted, and the first one, with the highest workload, is selected as the edge server. Then the neighbors of the selected edge server are defined. It includes all the nodes that their distance is less than the



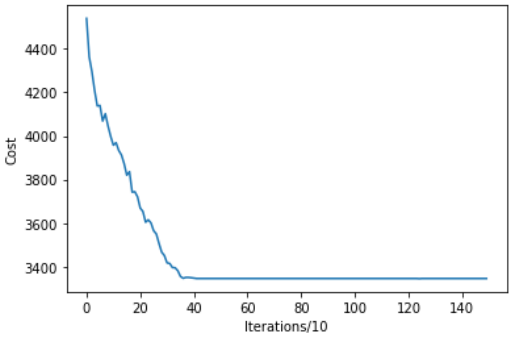
(a) 100 base stations



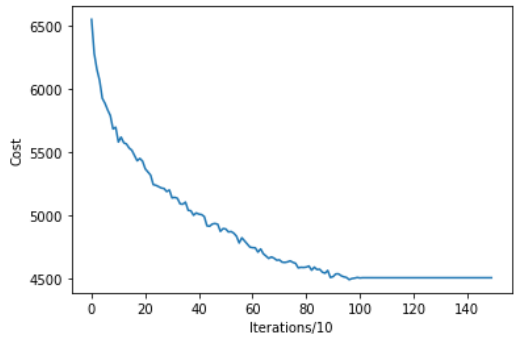
(b) 300 base stations



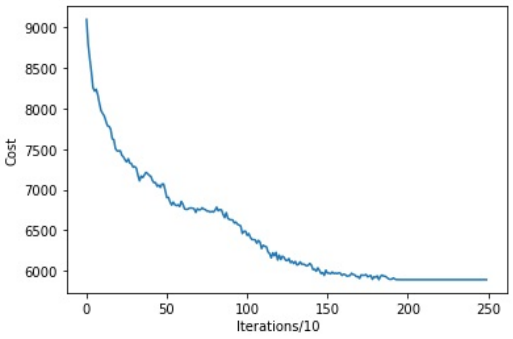
(c) 500 base stations



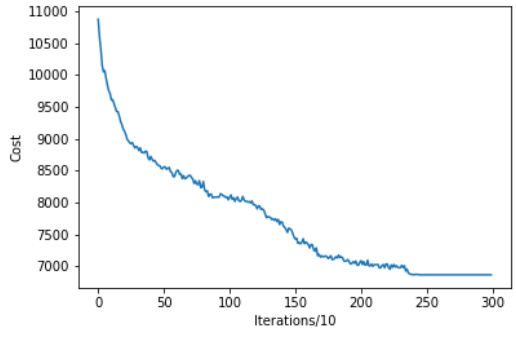
(d) 700 base stations



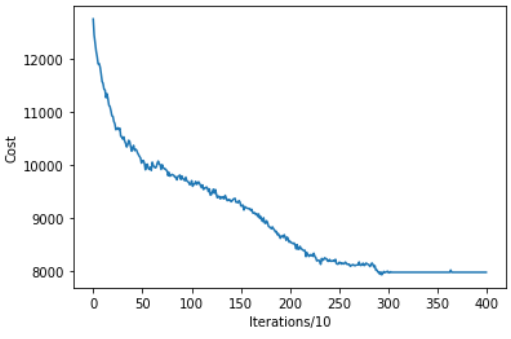
(e) 1000 base stations



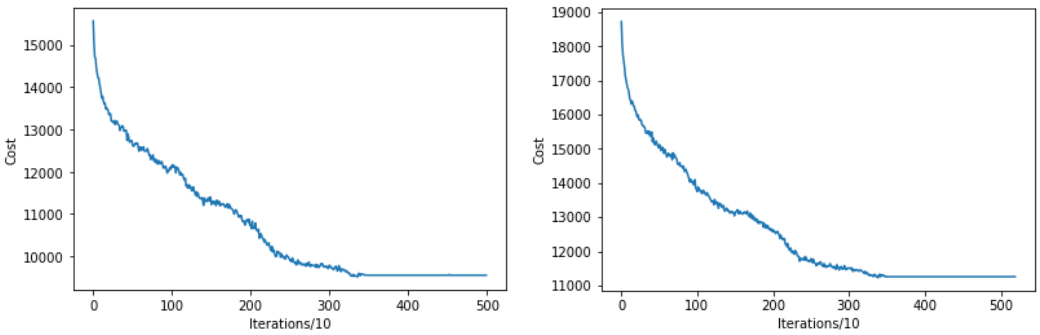
(f) 1400 base stations



(g) 1700 base stations



(h) 2000 base stations



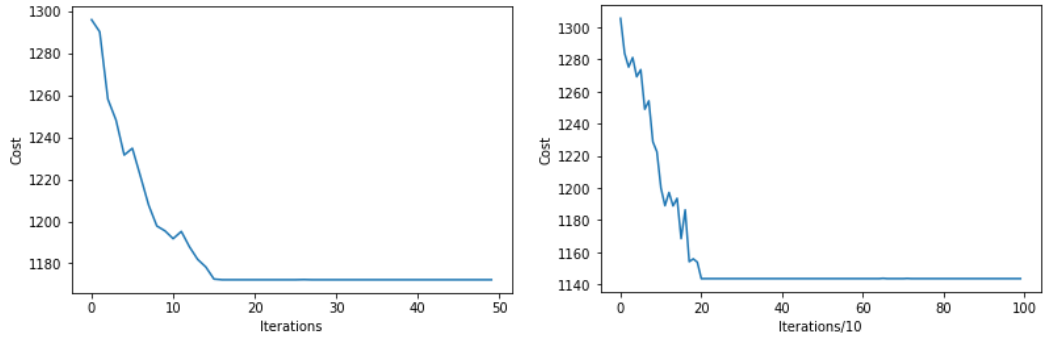
(i) 2400 base stations (j) 2798 base stations

Figure 4.2: Implementing Q-learning to find minimum edge servers for different number of base stations

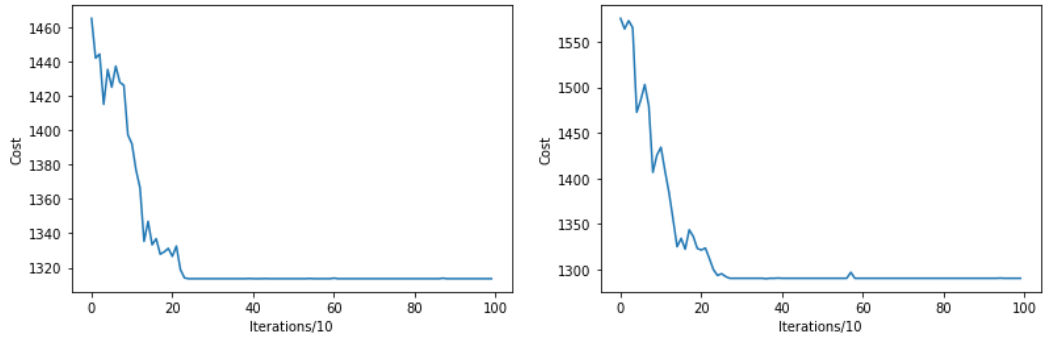
pre-defined threshold. The neighbor nodes offload their computation to the elected edge server until reaching the maximum capacity of the server. This cluster of the edge server and its connected base stations is removed from the initial set of base stations. Then the remaining nodes are again sorted based on their workloads, and the algorithm is repeated until all the base stations are assigned to an edge server.

The other two 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.

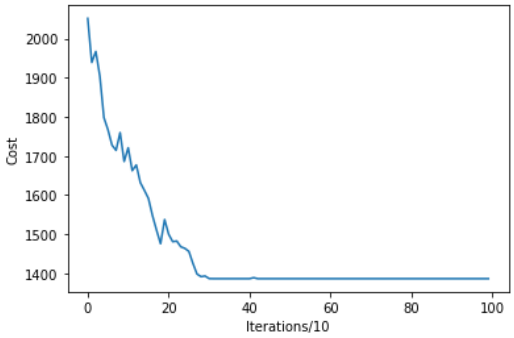
Figure. 4.5 and Table. 4.1 illustrate the performance of four algorithms for different number of base stations, from 100 to 2798. As the number of base stations increases (for more than 2000 base stations), the Top-DoF outperforms the Top-K (Fig. 4.5). The performance of the Random is interesting, and in most cases, it has some striking similarities to the performance of the Top-K in terms of network delay. Our Q-learning algorithm for having the minimum cost (QMC) has the best performance with considerably less delay (Fig. 4.5). The delay in our algorithm is, on average, 47.7% less than Top-K, 49.25% less than Top-DoF, and 47.64% less than Random.



(a) Distance threshold = $3km$ (b) Distance threshold = $5km$

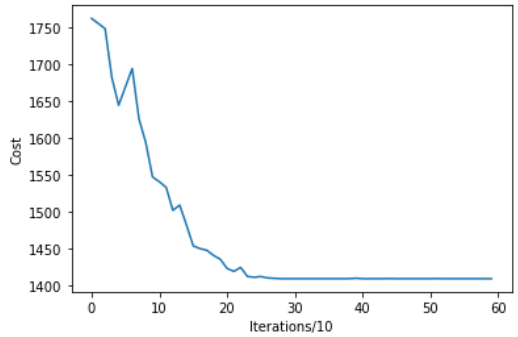


(c) Distance threshold = $7km$ (d) Distance threshold = $9km$

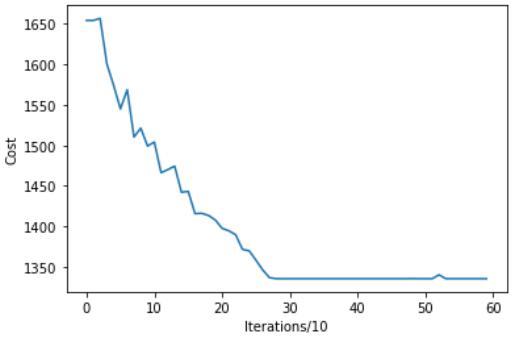


(e) Distance threshold = $11km$

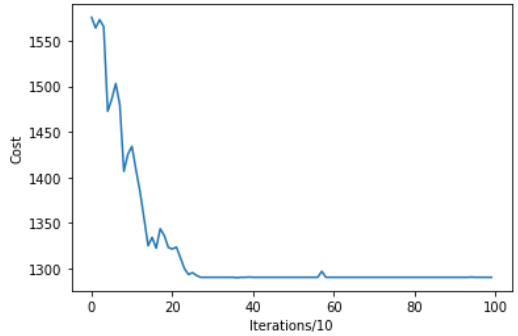
Figure 4.3: Performance of our proposed algorithm by considering different accepted network delays



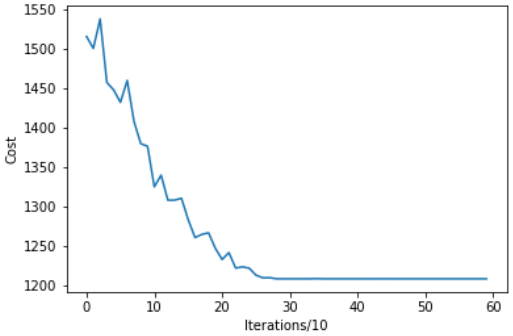
(a) Computation capacity = 100*requests/day*



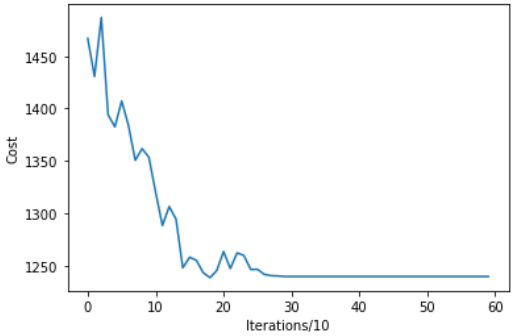
(b) Computation capacity = 120*requests/day*



(c) Computation capacity = 150*requests/day*



(d) Computation capacity = 170*requests/day*



(e) Computation capacity = 200*requests/day*

Figure 4.4: Performance of our proposed algorithm by considering different computation capacity for the edge servers

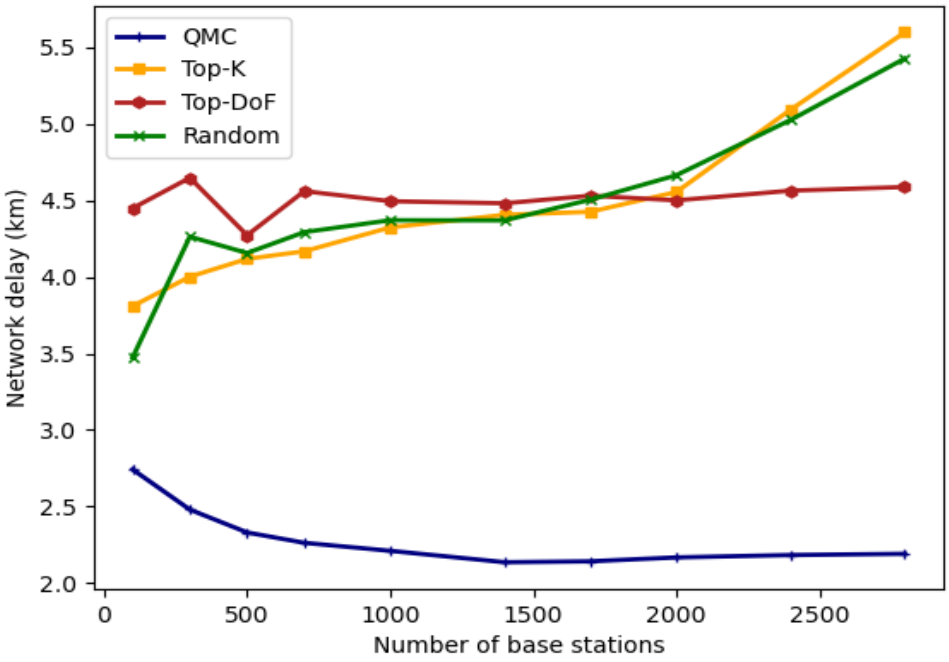


Figure 4.5: Results comparison in terms of network delay.

n	QMC	Top-K	Top-DoF	Random
100	**17**	19	18	18
300	**59**	63	63	64
500	**109**	114	112	114
700	**171**	176	172	175
1000	**224**	230	228	230
1400	**285**	290	288	292
1800	**318**	323	324	328
2000	**360**	363	364	365
2400	**428**	433	435	443
2798	**509**	512	517	521

Table 4.1: The number of selected edge servers with respect to the number of base stations

Table. 4.1 shows an approximately similar performance of diffident algorithms. However, in all cases, our algorithm has found fewer edge servers. In contrast to the algorithm's performance with respect to the network delay, concerning the number of edge servers, for more number of nodes, Top-K compared to Top-DoF has better results. It means that for 100 to 1400 base stations, Top-DoF has found fewer edge servers, while for 1800 to 2798 base stations, Top-K has better performance.

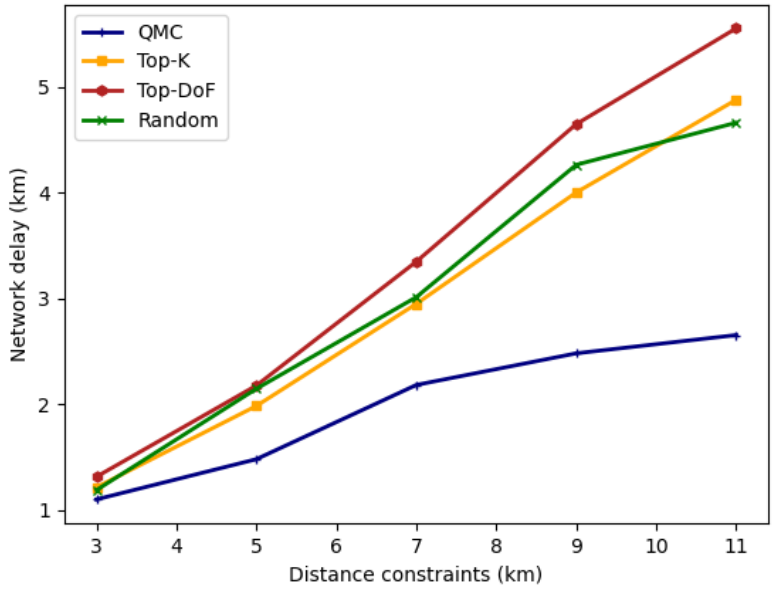
In the other experiment, the number of base stations is 300, and the maximum computation capacity of edge servers is 150 requests per day (Fig. 4.6). These two factors are fixed, and the distance constraint varies. The performance of four algorithms is compared in terms of the number of edge servers (K) in Figure. 4.6b, and in terms of network delay (D_N) in Figure. 4.6a, 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. However, the gap between our proposed algorithm and the others is transparent, especially when the distance threshold increases (Fig. 4.6a). The network delay of QMC is, on average, 28.88% less than Top-K, 36.38% less than Top-DoF, and 30.17% less than Random.

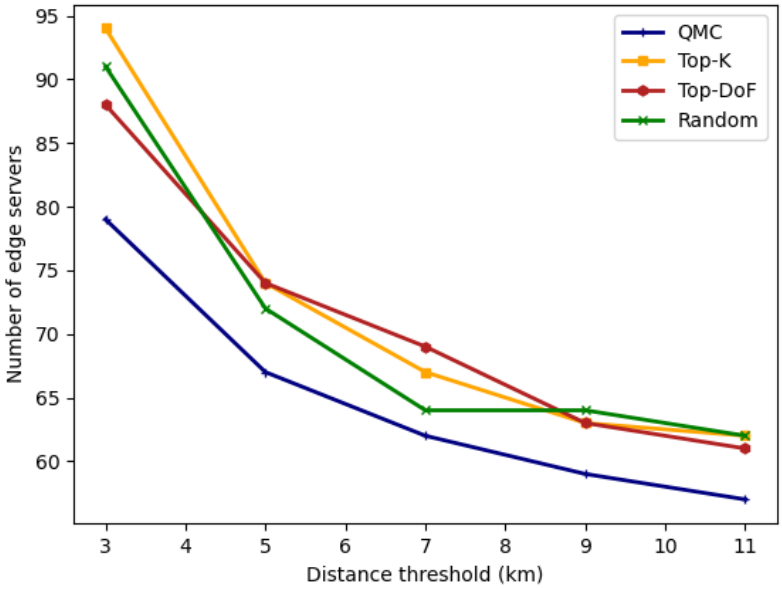
The number of selected edge servers decreases as the delay increases, except in one case for the Random algorithm. The random algorithm has exceptional results compared to Top-K and Top-DoF, and it outperforms in some situations (Fig. 4.6b).

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, as in the other experiments, our proposed algorithm noticeably outperforms the others benchmark approaches (Fig. 4.7). The Random algorithm in this experiment, on average, has a better performance compared to the Top-K and Top-DoF. To find the minimum number of edge servers, as the capacity increases, the number of edge servers decreases, and relatively similar performance for all the algorithms is observed (Table. 4.2). As the table shows, QMC has found more edge servers compared to the three other approaches for 100 nodes. The reason is in the penalty function's definition because it utilizes the same weights for the two objectives. As shown in Figure 4.7, the agent has acted in a way for receiving the



(a) Average network delay



(b) Number of selected edge servers

Figure 4.6: Comparison the four algorithms with respect to the distance constraint for 300 base stations

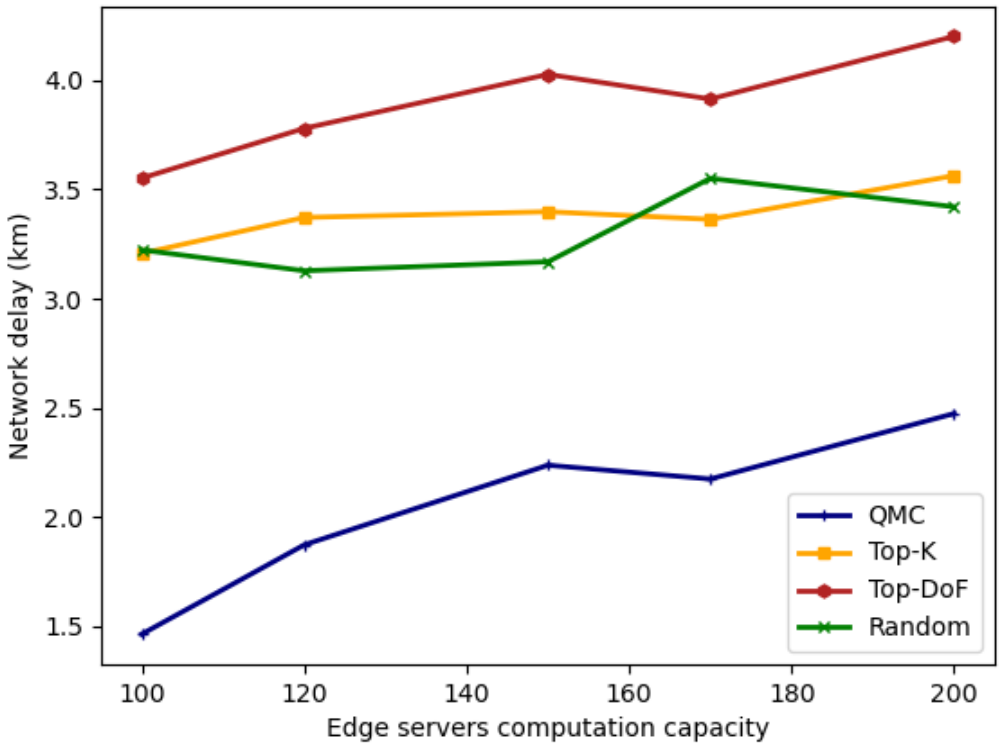


Figure 4.7: Average network delay with respect to the edge servers computation capacity

Computation capacity	QMC	Top-K	Top-DoF	Random
100	93	**91**	**91**	92
120	**75**	75	76	79
150	**59**	63	63	64
170	**54**	58	54	64
200	**47**	48	47	61

Table 4.2: The number of selected edge servers with respect to the edge servers computation capacity

minimum penalty in total. While it has found one or two more servers, the delay has significantly decreased. It is worth noting that in our algorithm, it is possible to add different weights for goals based on their importance or consider a higher value as the penalty of adding each edge server.

4.3 K-edge Servers Placement

4.3.1 Implementation Results

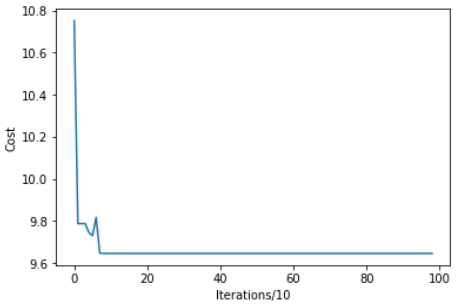
In this section, the aim is to demonstrate the performance of our two proposed algorithms when there is a fixed number of edge servers. The investigation for this problem is done for different numbers of base stations: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. These base stations are chosen randomly from those where latitude > 31.18 and longitude > 121.5, in order to consider a dense area.

Similar to [47], the number of edge servers is assumed as 1/10 number of base stations. However, this ratio is dependent on the available budget. Increasing the number of edge servers due to more budget usually results in better performance, especially from the delay point of view.

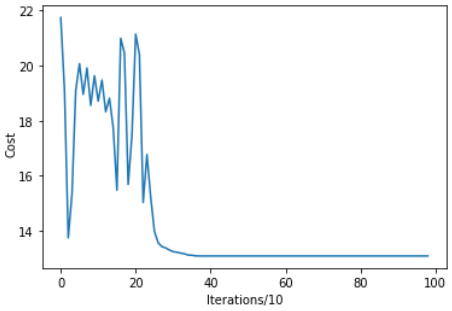
Figure. 4.8, and 4.9 demonstrate the convergence of the Q-learning for placement and optimal allocation for K-edge servers (QPAK) and the combination of Q-learning with K-means and Top-K (CQPAK) algorithms for the different number of base stations. CQPAK converges sooner compared to QPAK. For example, in Fig. 4.8j, selecting ten edge servers from 100 base stations QPAK needs about 13000 iterations to reach the near-optimal value, while the CQPAK converges before 8000 iterations (Fig. 4.9j). These results reveal the effect of action space on the computation time, and as the number of base stations increases, the difference becomes more evident.

4.3.2 Performance Evaluation

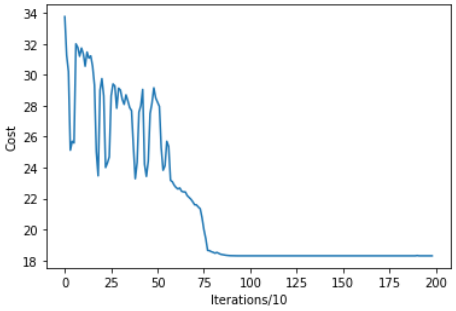
After validating the offered algorithms, their results are compared to the other benchmark methods. Based on the mentioned objectives in Eqs. 8, 9, and 10, three performance indicators are defined. For calculating the communication delay, the average



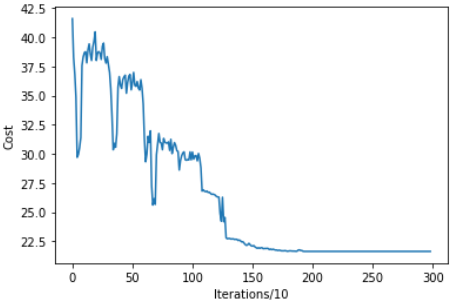
(a) Selecting 1 edge server.



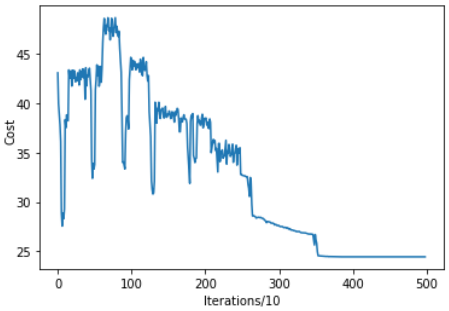
(b) Selecting 2 edge servers.



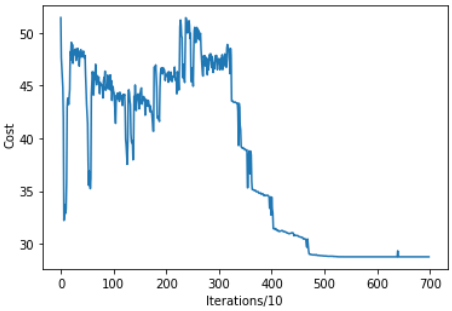
(c) Selecting 3 edge servers.



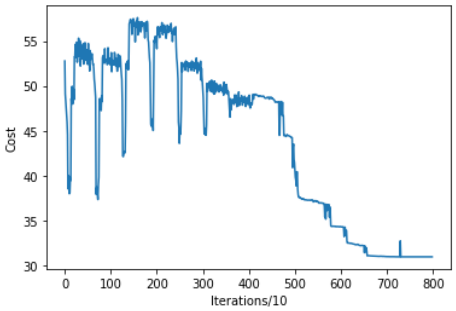
(d) Selecting 4 edge servers.



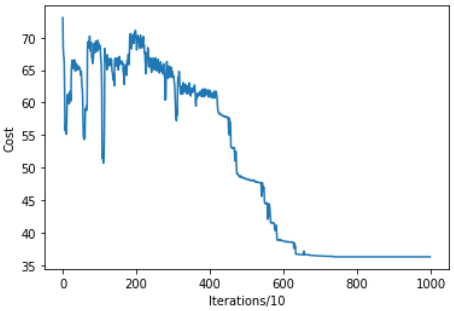
(e) Selecting 5 edge servers.



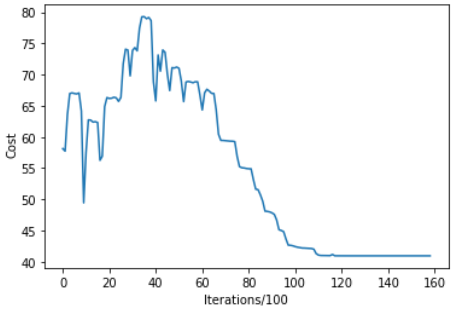
(f) Selecting 6 edge servers.



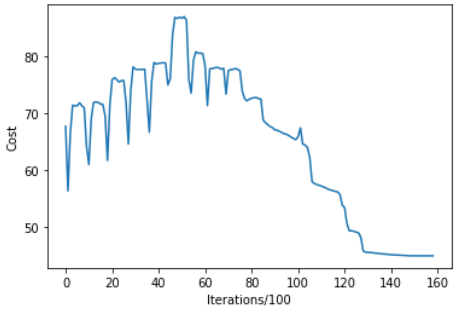
(g) Selecting 7 edge servers.



(h) Selecting 8 edge servers.

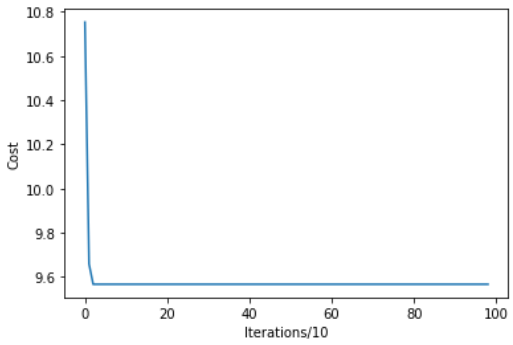


(i) Selecting 9 edge servers.

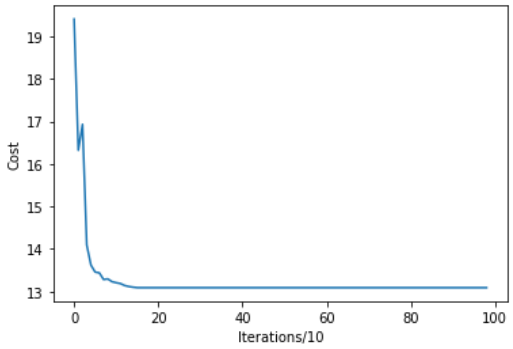


(j) Selecting 10 edge servers.

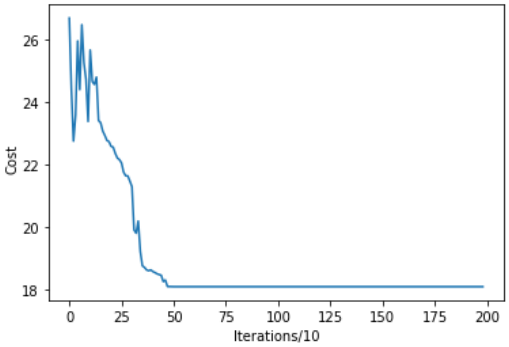
Figure 4.8: Implementing QPAK for the K-edge servers placement by considering different number of base stations.



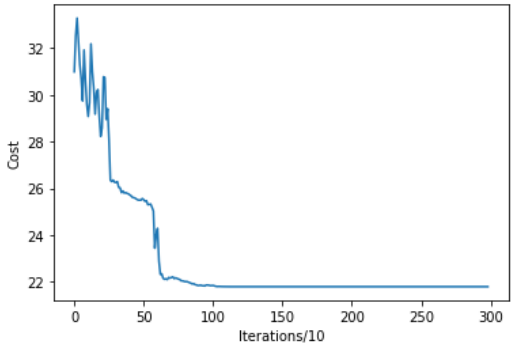
(a) Selecting 1 edge server.



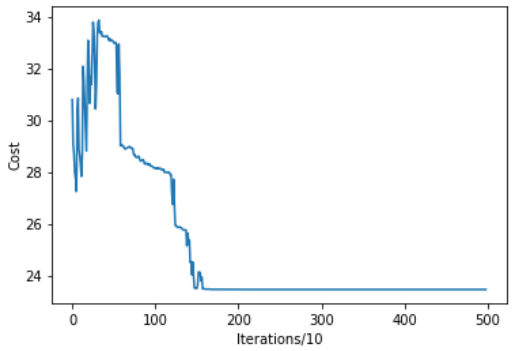
(b) Selecting 2 edge servers.



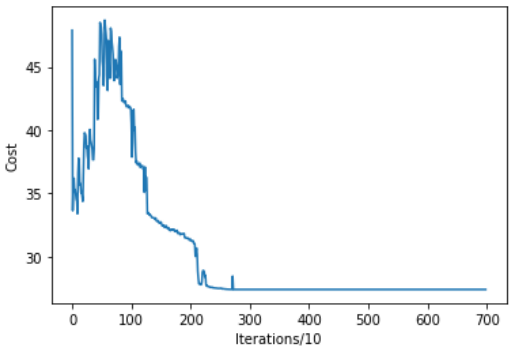
(c) Selecting 3 edge servers.



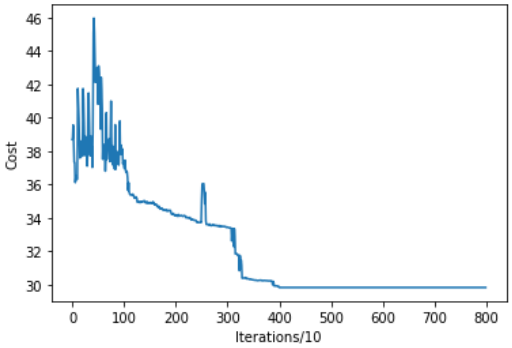
(d) Selecting 4 edge servers.



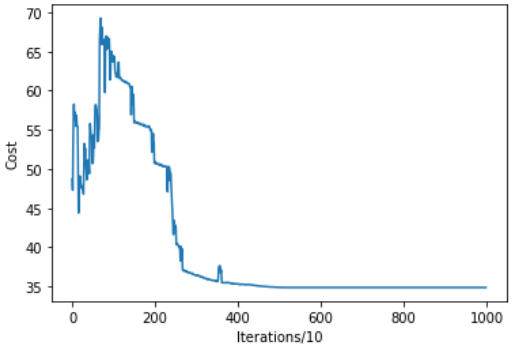
(e) Selecting 5 edge servers.



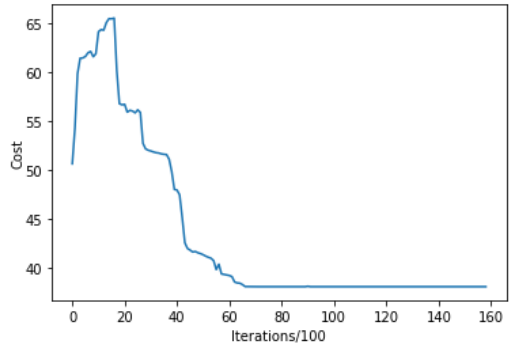
(f) Selecting 6 edge servers.



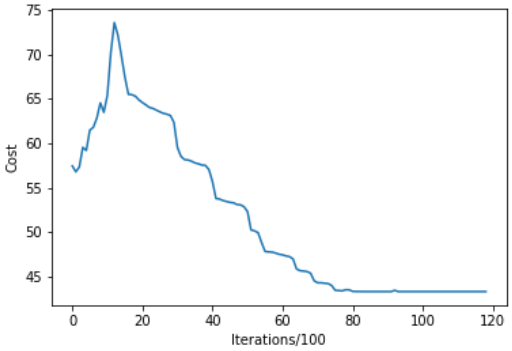
(g) Selecting 7 edge servers.



(h) Selecting 8 edge servers.



(i) Selecting 9 edge servers.



(j) Selecting 10 edge servers.

Figure 4.9: Implementing CQPAK for the K-edge servers placement by considering different number of base stations.

distance between base stations and their edge server is used as follows:

$$D_{communication} = \frac{\sum_{j=1}^{K}\sum_{i=1}^{n}(distance(bs_i, es_j) \times x_{ij})}{n} \tag{27}$$

where x_{ij} is a binary variable which shows if the base station is assigned to that edge server as follows:

$$x_{ij} = \begin{cases} 1, & \text{if } bs_i \text{ is connected to } es_j \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

Workload balancing is a factor for MEC performance evaluation [36]. To show the workload balancing the standard deviation was used. By assuming having K edge servers the workload balancing was calculated as follows:

$$WB = \sqrt{\frac{\sum_{j=1}^{K}(wl_j - \overline{wl}_{ES})^2}{K}} \tag{29}$$

$$\overline{wl}_{ES} = \frac{\sum_{i=1}^{n}\lambda_i}{K} \tag{30}$$

Eq. 30 is the average workload of all base stations over K edge servers. Eq. 29 computes how far we are from the ideal condition for workload balancing, so the smaller value shows the better offloading policy for a homogeneous network.

To show the fairness of our assignments in the network, for each cluster, the average delay same as the previous Eq. 27 is calculated called D_j. Then same as Eq. 29 the standard deviation is used for the cluster delays. The smaller value represents higher fairness. It means that users in different areas who are connected to the network are experiencing the same quality of service.

In addition to our two algorithms described in Chapter 3 for this problem, the K-means and Top-K are also used as base models to compare the performance of different approaches. K-means is one of the most common unsupervised clustering algorithms. For using this algorithm, K should be determined in advance, which is equal to the number of clusters. This algorithm partitions the dataset in a way

that within distance in each cluster is minimized while the distance between clusters is maximized. The K-means algorithm is sensitive to centroid initialization. K-means++ [3] is used as it applies a smarter initialization method. Fig. 4.10 illustrates the K-means clustering, based on the base stations' geographical location, which does not consider the workload of the base stations.

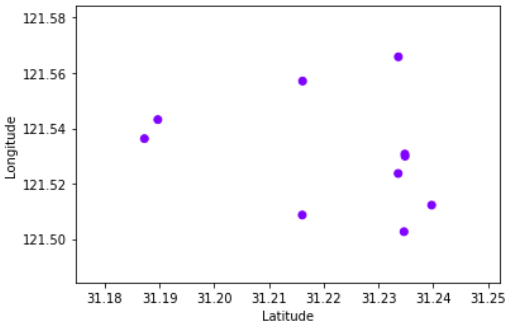
In Top-K, the K heaviest base stations in terms of the workload are selected as the edge servers. Then the other base stations are assigned to their nearest edge server until all of them are mapped to a particular edge server.

For comparing the performance of these four algorithms, three objectives were defined. One of the targets is to have a balanced network in terms of workload. As Fig. 4.11 represents CQPAK, QPAK have relatively similar performance better than the other two algorithms. In most cases, K-means has a more balanced workload compared to Top-K.

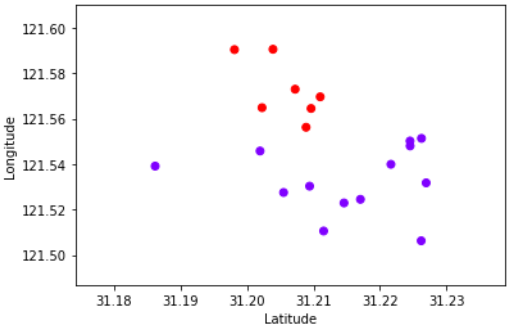
n	Top-K	K-means	QPAK	CQPAK
10	4.37	2.29	**2.24**	**2.24**
20	1.62	**1.49**	1.72	1.72
30	1.54	**1.15**	1.71	1.66
40	1.64	**1.45**	2.19	2.15
50	1.36	**0.82**	1.29	1.30
60	2.02	**0.94**	1.69	1.08
70	1.14	**0.87**	1.34	1.35
80	1.18	**0.77**	1.29	1.13
90	1.04	**0.78**	1.32	0.99
100	1.13	**0.66**	0.92	0.97

Table 4.3: The communication delay (km) (Eq.27) considering different number of base station

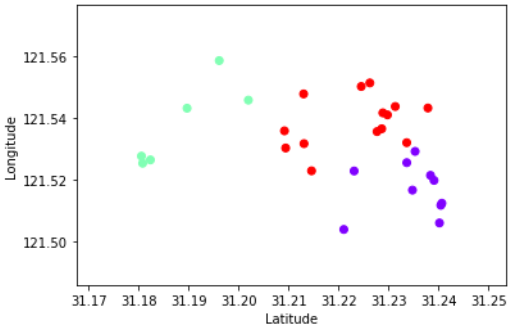
Table. 4.3 shows the communication delay where K-means outperforms the other models, except the first row, where the number of base stations is equal to 10. First, the reason for the better performance of K-means is that our models are designed to find the optimal locations and allocations based on the trade-off among three objectives, not just one. Second, the reason for the better performance of our models



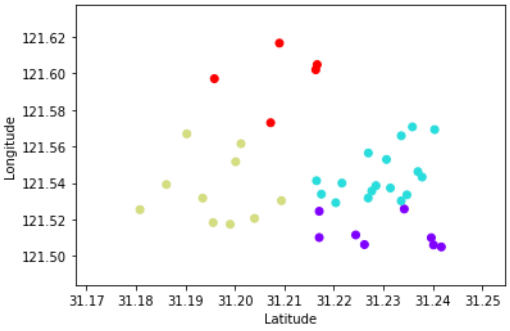
(a) Selecting 1 edge server.



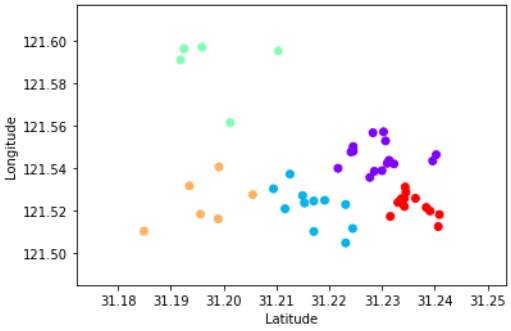
(b) Selecting 2 edge servers.



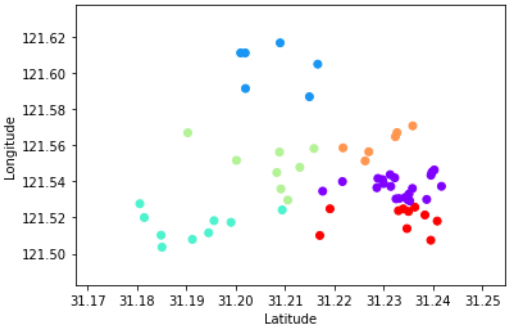
(c) Selecting 3 edge servers.



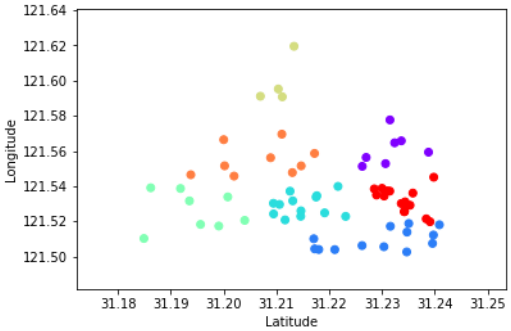
(d) Selecting 4 edge servers.



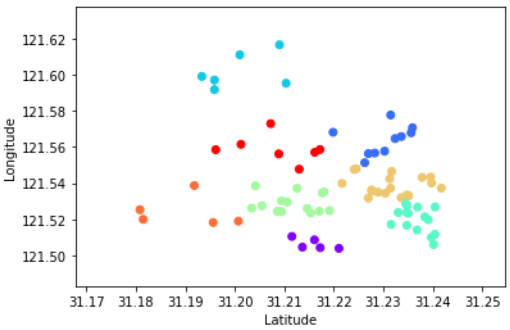
(e) Selecting 5 edge servers.



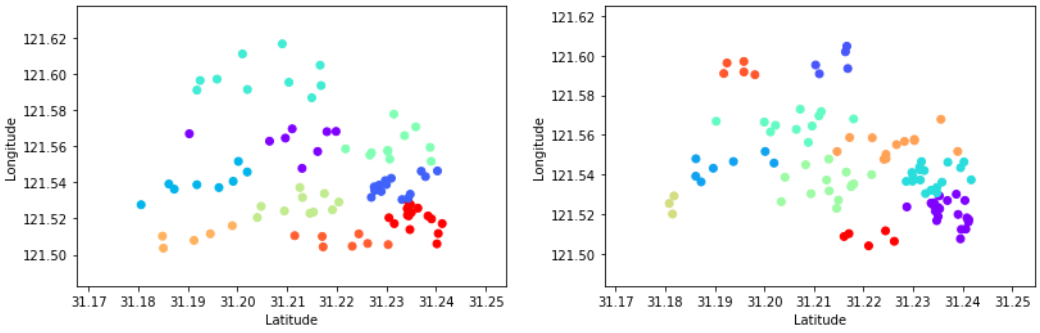
(f) Selecting 6 edge servers.



(g) Selecting 7 edge servers.



(h) Selecting 8 edge servers.



(i) Selecting 9 edge servers.

(j) Selecting 10 edge servers.

Figure 4.10: Implementing K-means for the K-edge servers placement by considering different number of base stations.

when there is just one edge server/cluster is that the agent focuses only on minimizing the communication delay because the penalty value for the other two objectives is similar in all iterations. The edge server placement and base stations' allocation for these ten nodes is graphically illustrated in Fig. 4.12. Unlike our methods and Top-K, as can be seen in Fig 4.12a, K-means adds a new location for the placement of the edge server. It is worth mentioning that K-means could perform better than our methods in some other random ten node selections due to the added node that might reduce the delay. However, based on our different experiments, in most cases, QPAK and CQPAK have better results. Therefore, the result in this specific case where only one edge server is considered is highly dependent on the nodes' placement.

Although Table. 4.4 shows that K-means works well in creating the fair clusters, this performance is dependent on the nodes' locations and their closeness. For example, for 20 and 30 nodes, the other algorithms have better results.

After scaling all the values for different parts of the objectives, the sum of these values is calculated. This sum shows the cost of each decision based on the three goals. Fig. 4.13 demonstrates the performance of the four mentioned methods by considering the different number of base stations when the number of edge servers is fixed, and it is equal to 1/10 of the total number of base stations.

Our algorithms outperform the other two algorithms (Fig. 4.13). As we mentioned in Chapter 3, using CQPAK might improve the performance because it reduces the action space, for example, for 40 to 80 nodes. On the other hand, it might affect the performance as the agent cannot have enough explorations, and it might pass over

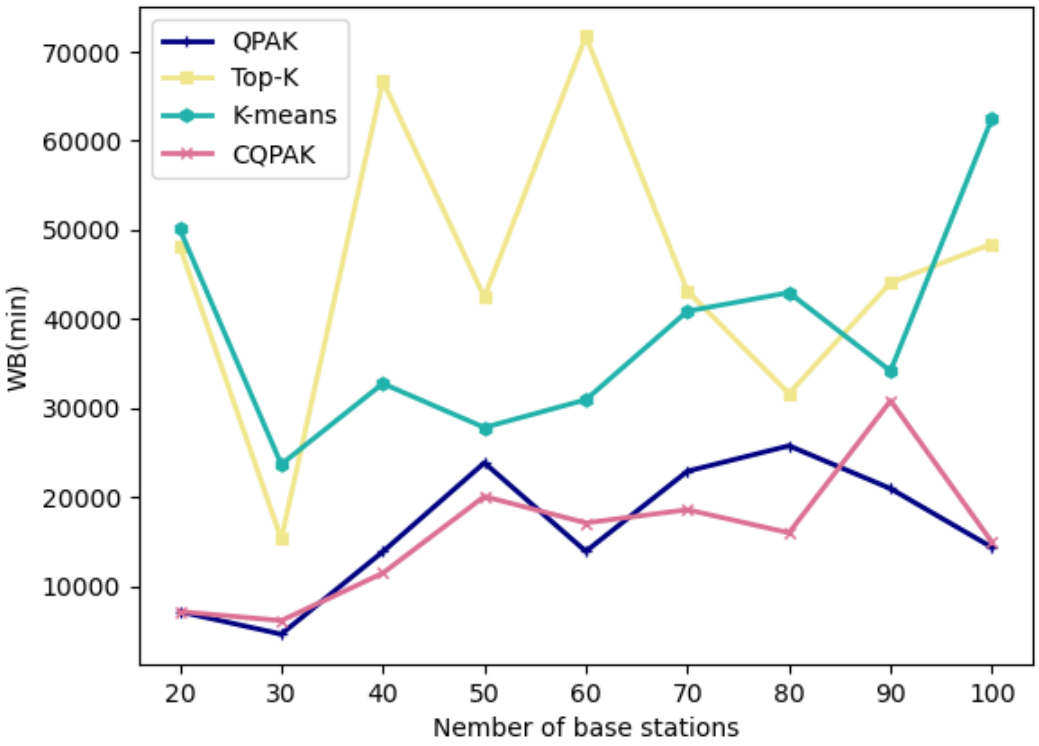
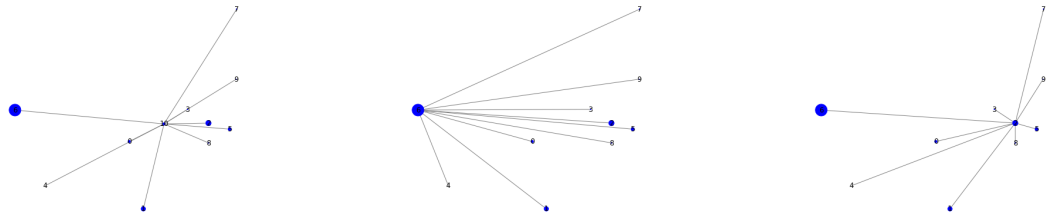


Figure 4.11: The workload balancing (*min*) (Eq. 29) of the edge servers considering different number of base station.



(a) K-means. (b) Top-K (c) QPAK / CQPAK.

Figure 4.12: Edge server placement and base station allocation for $n = 10$.

n	Top-K	K-means	QPAK	CQPAK
20	0.074	0.293	**0.028**	**0.028**
30	0.119	0.191	0.065	**0.060**
40	0.564	**0.192**	0.735	0.502
50	0.631	**0.290**	0.430	0.580
60	0.878	**0.191**	0.607	0.404
70	0.436	**0.181**	0.384	0.445
80	0.295	**0.225**	0.654	0.435
90	0.376	**0.217**	0.347	0.329
100	0.580	**0.166**	0.207	0.292

Table 4.4: The fairness (km) of the network considering different number of base station

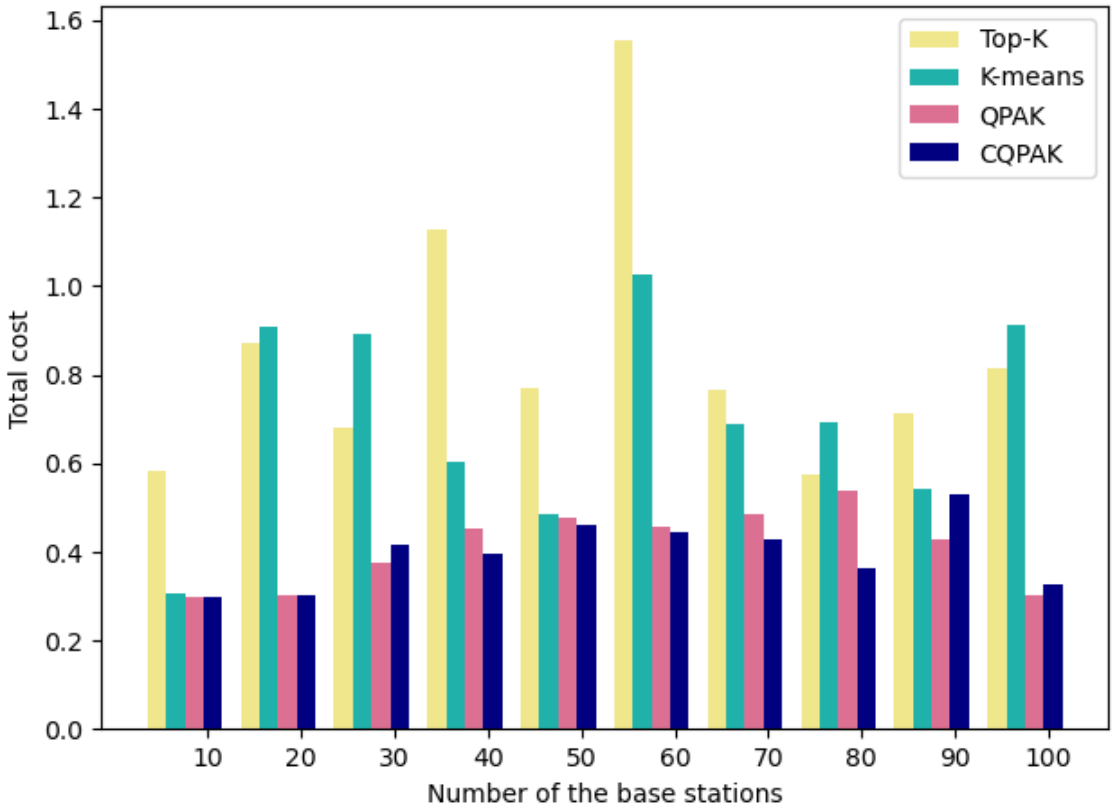
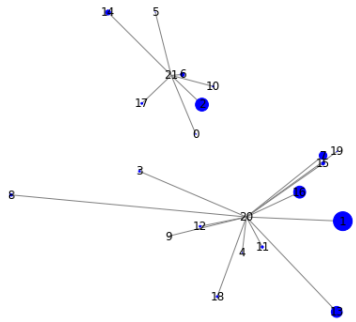
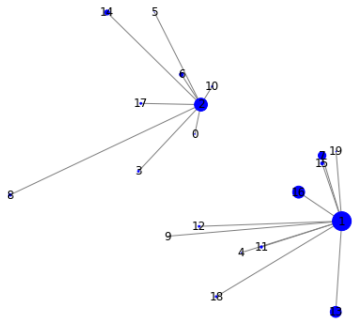


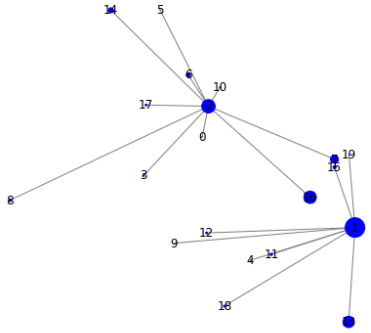
Figure 4.13: The total cost value for the four algorithms respect to the number of the base stations.



(a) K-means.



(b) Top-K.



(c) QPAK and CQPAK.

Figure 4.14: Comparing the placement and allocation of the four algorithms for 20 base stations.

some better decisions like for 30, 90, and 100 nodes. In total, these two algorithms have relatively similar performance, while CQPAK needs less computation power and time. The performance of Top-K and K-means varies for the different number of nodes based on the workload distribution and nodes' location.

The detailed performance of our proposed algorithms, in comparison to the others, is illustrated in Fig. 4.14, which is a graph view for 20 base stations. The size of the nodes represents their scaled workload. In Fig. 4.14a nodes 20 and 21 are the locations that are added to the network by K-means for the edge server placement. As can be seen, the workloads are not considered because the K-means partitions the nodes based on their geographical location. Although for this experiment, in Top-K and our

models, the opted nodes for the edge server placement are the alike (Fig. 4.14b, 4.14c), the results are dissimilar because of the different allocations. By considering all three objectives, our model has mapped nodes 7 and 16 to node 2. However, this decision results in a more average communication delay but increases the load balancing that consequently decreases the delay caused by the queue. Besides, it increases fairness in clustering, which is the third objective that needs to be considered. The above example shows how our agent is trying to create a balance between objectives.

4.4 Summary

In this chapter, we demonstrated the convergence of our proposed RL-frameworks for the two problems we address in this thesis: minimizing the cost of MEC design and finding the optimal placement of the K-edge servers by considering different conditions. For each of our two models, the performance indicators, based on the objectives, were defined to evaluate different approaches. Our algorithms showed better results compared to other benchmark methods when all the goals are considered jointly. Moreover, while the total performance of Top-K, top-DoF, Random, and K-means varies based on the base stations' location and nodes' workload, our RL-agent can find the trade-off by discovering where improvements are needed in order to have a better total performance for all different situations. Therefore, based on the obtained results, in the light of efficient penalty function and precise state-space, our RL-based frameworks are able to handle multi-objective problems even for large-scale problems.

Chapter 5

Conclusion and Future Work

MEC is a computing paradigm that is highly promising for the emerging 5G applications that require real-time response. Defining the optimal placement for edge servers and optimally assigning the users or base stations to the edge servers are key points toward a better performance of the network. After assessing the literature review and witnessing the promising results of RL for the placement optimization, in this study, the main goal was to define an efficient RL-framework for this joint problem.

Therefore, in this thesis, we developed new RL-based algorithms by adapting the classic Q-learning for jointly optimizing the placement and computation offloading. We considered two key perspectives, minimizing the cost and finding the optimal MEC design with respect to placement and allocation for K-edge servers. In addition to having convergence for our proposed algorithms, which is a difficult challenge in RL, we have obtained promising results demonstrating that RL-based solutions have better performance compared to other benchmark methods.

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 multi-objective problems, lack of a specific static end goal and curse of dimensionality caused by massive action space and state-space explosion. Our solutions for this specific joint problem and for these mentioned issues are explained. Despite the mentioned challenges, the key to Q-learning application for this large scale real-world problem is a precise and efficient definition of the state space, action space, and penalty function.

In the first problem, to find the optimal design with the minimum cost, the constraints are added to the algorithm as rules for limiting the branching, and then a new step is considered for offering the possible actions based on the previous decisions in each episode. In the second problem, K-edge server placement, after having an efficient Q-learning based algorithm, for reducing the computation time, we offered the combination of K-means and top-K with Q-learning.

In summary, we have been able to model the joint problem of the placement and base station allocation as an RL problem solvable by employing the Q-table. As future work, we will investigate the use of deep RL algorithms for the second model and analyse the convergence, complexity, and efficiency issues. The other way to extend this work is to eliminate the assumption that each base station can only be connected to one edge server and consider the situation where the edge servers are shareable. Each edge server can be considered as a smart agent that communicates with the other servers. This becomes a multi-agent game where the edge servers can collaborate with each other to increase the performance.

One limitation of our algorithms is that the agent needs to perform the exploration process repeatedly and needs to be retrained for every new placement problem including increasing the number of base stations or changing the constraints. The ability of generalizing the algorithms for larger and unseen networks might be achieved by defining the MEC as a graph and then deploying and training a graph neural network to represent the state space.

Besides setting the rules and using the information in the state definition for reducing the action space, the next step can be training a neural network and using it for classifying different actions, which might increase the efficiency of the RL algorithms to reach better results. As we are dealing with a sequential decision-making problem, a recurrent neural network can also be used to help the RL-agent take better actions.

Bibliography

[1] Ejaz Ahmed, Abdullah Gani, Mehdi Sookhak, Siti Hafizah Ab Hamid, and Feng Xia. Application optimization in mobile cloud computing: Motivation, taxonomies, and open challenges. *Journal of Network and Computer Applications*, 52:52–68, 2015.

[2] 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*, 8:54074–54084, 2020.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, 2006.

[4] Paramvir Bahl, Richard Y. Han, Li Erran Li, and Mahadev Satyanarayanan. Advancing the state of mobile cloud computing. In *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, page 21–28, New York, NY, USA, 2012. Association for Computing Machinery.

[5] Craig Boutilier, Alon Cohen, Amit Daniely, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. Planning and learning with stochastic action sets. *arXiv preprint arXiv:1805.02363*, 2018.

[6] Kun Cao, Liying Li, Yangguang Cui, Tongquan Wei, and Shiyan Hu. Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing. *IEEE Transactions on Industrial Informatics*, PP:1–1, 02 2020.

[7] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Pr'emont-Schwarz, and André Ciré. Combining reinforcement learning and constraint programming for combinatorial optimization. *ArXiv*, abs/2006.01610, 2020.

[8] Yash Chandak, Georgios Theocharous, Blossom Metevier, and Philip S Thomas. Reinforcement learning when all actions are not always available.

[9] Meng-Hsi Chen, Ben Liang, and Min Dong. A semidefinite relaxation approach to mobile cloud offloading with computing access point. In *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 186–190. IEEE, 2015.

[10] Boutheina Dab, Nadjib Aitsaadi, and Rami Langar. Q-learning algorithm for joint computation offloading and resource allocation in edge cloud. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 45–52. IEEE, 2019.

[11] K. Dolui and S. K. Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, 2017.

[12] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, B. Dietrich, David P. Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. *ArXiv*, abs/2006.03750, 2020.

[13] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[14] Anna Goldie and Azalia Mirhoseini. Placement optimization with deep reinforcement learning. In *Proceedings of the 2020 International Symposium on Physical Design*, pages 3–7, 2020.

[15] Yan Guo, Shangguang Wang, Ao Zhou, Jinliang Xu, Jie Yuan, and Ching-Hsien Hsu. User allocation-aware edge cloud placement in mobile edge computing. *Software: Practice and Experience*, 50(5):489–502, 2020.

[16] Ahmad Hammoud, Hani Sami, Azzam Mourad, Hadi Otrok, Rabeb Mizouni, and Jamal Bentahar. AI, blockchain, and vehicular edge computing for smart and secure IoV: Challenges and directions. *IEEE Internet of Things Magazine*, 3(2):68–73, 2020.

[17] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5G.

[18] Liang Huang, Xu Feng, Cheng Zhang, Liping Qian, and Yuan Wu. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks*, 5(1):10–17, 2019.

[19] Mike Jia, Jiannong Cao, and Weifa Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Transactions on Cloud Computing*, 5(4):725–737, 2015.

[20] M. Kamoun, W. Labidi, and M. Sarkiss. Joint resource allocation and offloading strategies in cloud enabled cellular networks. In *2015 IEEE International Conference on Communications (ICC)*, pages 5529–5534, 2015.

[21] R. Kemp, N.O. Palmer, T. Kielmann, and H.E. Bal. Cuckoo: a computation offloading framework for smartphones. In *2nd Int. Conf. on Mobile Computing, Applications, and Services (MobiCASE 2010)*, 2010.

[22] S. Lee, S. Lee, and M. K. Shin. Low cost MEC server placement and association in 5G networks. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 879–882, 2019.

[23] Y. Li and S. Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 66–73, 2018.

[24] Yuanzhe Li and Shangguang Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 66–73. IEEE, 2018.

[25] Juan Liu, Yuyi Mao, Jun Zhang, and Khaled B Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1451–1455. IEEE, 2016.

[26] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[27] Nina Mazyavkina, Sergey I. Sviridov, Sergei V. Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *ArXiv*, abs/2003.03600, 2020.

[28] J. Meng, C. Zeng, H. Tan, Z. Li, B. Li, and X. Li. Joint heterogeneous server placement and application configuration in edge computing. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 488–497, 2019.

[29] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[30] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020.

[31] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. *arXiv preprint arXiv:1706.04972*, 2017.

[32] A. Mukherjee, D. De, and D. G. Roy. A power and latency aware cloudlet selection strategy for multi-cloudlet environment. *IEEE Transactions on Cloud Computing*, 7(1):141–154, 2019.

[33] Kevin E Murray and Vaughn Betz. Adaptive fpga placement optimization via reinforcement learning.

[34] Dilay Parmar, A. Kumar, Ashwin Nivangune, P. Joshi, and U. Rao. Discovery and selection mechanism of cloudlets in a decentralized MCC environment. *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 15–16, 2016.

[35] 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*, 8:34313–34325, 2020.

[36] Kai Peng, Xiaolong Xu, Lixin Zheng, Jiabin Wang, and Qingjia Huang. A survey on mobile edge computing: Focusing on service adoption and provision. *Wireless Communications and Mobile Computing*, 2018, 10 2018.

[37] Gaith Rjoub, Jamal Bentahar, and Omar Abdel Wahab. Bigtrustscheduling: Trust-aware big data task scheduling approach in cloud computing environments. *Future Gener. Comput. Syst.*, 110:1079–1097, 2020.

[38] Gaith Rjoub, Omar Abdel Wahab, Jamal Bentahar, and Ahmed Saleh Bataineh. A trust and energy-aware double deep reinforcement learning scheduling strategy for federated learning on IoT devices. In Eleanna Kafeza, Boualem Benatallah, Fabio Martinelli, Hakim Hacid, Athman Bouguettaya, and Hamid Motahari, editors, *Service-Oriented Computing - 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14-17, 2020, Proceedings*, volume 12571 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2020.

[39] Hani Sami, Azzam Mourad, Hadi Otrok, and Jamal Bentahar. Fscaler: Automatic resource scaling of containers in fog clusters using reinforcement learning. In *16th International Wireless Communications and Mobile Computing Conference, IWCMC 2020, Limassol, Cyprus, June 15-19, 2020*, pages 1824–1829. IEEE, 2020.

[40] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[41] Mahadev Satyanarayanan. Mobile computing: The next decade. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, New York, NY, USA, 2010. Association for Computing Machinery.

[42] Tanmoy Sen and Haiying Shen. Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems. In *2019 IEEE*

3rd International Conference on Fog and Edge Computing (ICFEC), pages 1–10. IEEE, 2019.

[43] Weisong Shi, George Pallis, and Zhiwei Xu. Edge computing [scanning the issue]. *Proceedings of the IEEE*, 107(8):1474–1481, 2019.

[44] Roger W Sinnott. Virtues of the haversine. *S&T*, 68(2):158, 1984.

[45] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[46] Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, Mohammad Alizadeh, et al. Learning generalizable device placement algorithms for distributed machine learning. In *Advances in Neural Information Processing Systems*, pages 3981–3991, 2019.

[47] Shangguang Wang, Yan Guo, Ning Zhang, Peng Yang, Ao Zhou, and Xuemin Sherman Shen. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Transactions on Mobile Computing*, 2019.

[48] Shangguang Wang, Yali Zhao, Lin Huang, Jinliang Xu, and Ching-Hsien Hsu. Qos prediction for service recommendations in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:134–144, 2019.

[49] Shangguang Wang, Yali Zhao, Jinlinag Xu, Jie Yuan, and Ching-Hsien Hsu. Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:160–168, 2019.

[50] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[51] Yifei Wei, Zhaoying Wang, Da Guo, and F. Yu. Deep q-learning based computation offloading strategy for mobile edge computing. *Computers, Materials & Continua*, 59:89–104, 01 2019.

[52] Yuan Zhang, Hao Liu, Lei Jiao, and Xiaoming Fu. To offload or not to offload: An efficient code partition algorithm for mobile cloud computing. In *2012 IEEE*

1st International Conference on Cloud Networking (CLOUDNET), pages 80–86, 2012.

[53] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mob. Networks Appl.*, 16(3):270–284, 2011.

[54] Rui Zhao, Xinjie Wang, Junjuan Xia, and Liseng Fan. Deep reinforcement learning based mobile edge computing for intelligent internet of things. *Physical Communication*, 43:101184, 2020.