Minimizing Data Retrieval Delay in Edge Computing

Kolichala Rajashekar[1(✉)], Souradyuti Paul[1], Sushanta Karmakar[2], and Subhajit Sidhanta[1]

[1] Indian Institute of Technology Bhilai, Bhilai, India
kolichalar@iitbhilai.ac.in
[2] Indian Institute of Technology Guwahati, Guwahati, India

Abstract. For real-time mission-critical applications such as forest fire detection, oil refinery monitoring, etc., the edge computing paradigm is heavily used to process data fetched from IoT devices spread over a considerably large geographical region. For such real-time edge computing applications working under stringent deadlines, the overall *retrieval delay*, i.e., the delay in fetching the data from the IoT devices to the edge servers, needs to be minimized; Otherwise, the retrieval delay in fetching the data from IoT devices distributed over such a large geographical region can be prohibitively large. To achieve the above goal, each IoT device must be assigned to a particular edge server while considering the relative positioning as per the topology of the edge cluster. We prove that the above assignment of IoT devices to an edge cluster, which we denote as the Edge Assignment Problem (EAP), is NP-Hard. Therefore, obtaining a polynomial time solution is infeasible. For the above EAP problem, instead of performing both exploration and exploitation on the search space, state-of-the-art heuristic algorithms will only exploit the search space. As a result, these algorithms are unable to achieve an appreciably large reduction in the overall retrieval delay.

To that end, we propose a Deep Reinforcement Learning-based algorithm that is able to produce a near-optimal assignment of IoT devices to the edge cluster while ensuring that none of the edge servers is overloaded. We motivate and demonstrate our proposed algorithm with the use case of federated learning (FL) - a popular distributed machine learning paradigm that is based on the principle of edge computing such that the clients, i.e., edge servers, train local models from the data obtained from local IoT devices. These local models are further aggregated into a global model at an aggregator (the cloud/fog) by exchanging the model parameters instead of raw data. In that case, an optimal assignment of the IoT devices to each edge device is necessary for reducing the training time for the local models, which in turn, results in reducing the overall delay of federated learning. Using experiments that emulate real-world deployment scenarios, we demonstrate that our algorithm outperforms the state of the art in reducing the retrieval delay in the general edge computing scenario, while also minimizing the local training time in federated learning.

Keywords: IoT · edge computing · reinforcement learning

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 594, pp. 63–85, 2024.
https://doi.org/10.1007/978-3-031-63992-0_5

1 Introduction

We consider real-time edge computing applications, such as traffic monitoring (i.e., the detection of traffic violations), autonomous driving, oil refinery monitoring (i.e., an inspection of fluid levels in oil tanks), forest fire monitoring, etc. [23,37,43]. These applications are usually executed under stringent deadlines specified in the service level agreement (SLAs), for example, roughly 3 s in certain oil refinery monitoring use cases [53]. Typically these applications accept input data generated by IoT devices, such as wireless sensors. The above data is then processed by a cluster of edge servers performing some distributed real-time analytics tasks. From the above group of IoT devices, one or more devices are *assigned* to an edge server. Each edge server in the cluster collects and preprocesses a portion of the data [34]. Minimizing the *retrieval delay*[1] in fetching the data from the IoT devices results in minimizing the overall execution time of a real-time edge computing application. This minimization of retrieval delay will enable the application to satisfy its real-time requirement. Depending on the relative positions of IoT devices in a large geographical region and the topology of the edge cluster, the retrieval delay involved may vary. We consider a distributed deployment where the data from the IoT devices are retrieved to the edge servers in the cluster parallel. Hence, the overall retrieval delay for fetching the data is given as the maximum retrieval delay for fetching data from individual IoT devices. Our goal is to minimize this overall retrieval delay experienced by the end user, which we hereon denote as the retrieval delay. The growing number of IoT devices, in turn, will cause a rapid rise in the overall retrieval delay [10]. Hence, to enable real-time performance, the IoT devices must be assigned to individual edge servers in a topology-aware manner. This, in turn, minimizes the retrieval delay.

The above goal of minimizing the retrieval delay is particularly important for federated learning since the local training time, i.e., the time taken for training the local models, is significantly impacted by the retrieval delay. We empirically prove this later in the Evaluation Section of the paper. Federated learning has steadily risen to the ranks of one of the most widely adopted ML approaches because of its unique ability to circumvent data privacy issues by training on local datasets, as well as its obvious performance scalability due to inherent distributed and decentralized architecture. Federated learning is now becoming a "go-to" choice for running distributed machine learning workloads on an edge computing infrastructure, where the edge servers act as clients that train the local models using the data collected from the IoT devices in its proximity [50]. Though client (i.e., edge device) selection is a widely studied topic, the prior literature has not addressed the issue of appropriate assignment of IoT devices to each edge server, which would affect the local training time significantly.

[1] We consider the retrieval delay as the delay in fetching the data at an edge server from its corresponding IoT device.

To that end, this paper addresses the problem of an *edge assignment*[2], i.e., configuration of a given edge cluster in such a way that the IoT devices are assigned to individual edge servers such that the retrieval delay is minimized. At the same time, our proposed algorithm must not overload any edge server in the cluster by assigning too many IoT devices to it. This is because the edge server is resource-constrained and can be easily overloaded by workload from multiple IoT devices [44]. In this paper, we define an *optimal* edge assignment, which results in the minimization of the retrieval delay while ensuring that none of the edge servers is overloaded.

We formulate the above problem as a version of the bottleneck assignment problem which we denote as the *Edge Assignment Problem* (EAP). We prove that the *EAP* is NP-Hard in nature. Though, to the best of our knowledge, this exact problem has not been addressed in the prior literature, researchers have proposed various heuristics to obtain a near-optimal solution for similar optimization problems [14].

For a smaller topology, i.e., a network topology comprising a smaller set of IoT and edge servers, state-of-the-art tools such as Google OR-Tool [9] can solve the problem in a reasonable amount of time. If the number of devices is considerably large to the extent thousands of devices spread out across a very large region in a realistic scenario such as an oil refinery monitoring, we demonstrate that such solvers take a much longer time to reach a solution [33], Further, several heuristic algorithms have been proposed to solve different variations of the assignment problem, but some of the algorithms get stuck at local optima [16,27]. There are also several prior works which have focused on the edge server selection with respect to the computational capability of an edge server [22].

In that case, *reinforcement learning* (RL) is used to obtain a near-optimal solution to the assignment problem [32,36]. Following their lead, we also leverage reinforcement learning approaches, specifically *Deep−Q learning* [46] to provide a near-optimal solution to EAP. The contributions of the paper are as follows.

- We propose an algorithm that provides a near-optimal solution to the edge assignment problem, assigning a group of IoT devices to the edge cluster while ensuring that none of the edge servers is overloaded.
- Using experiments that emulate real-world deployment scenarios, we demonstrate that our algorithm outperforms the baseline approaches in minimizing the retrieval delay.
- We also show that the assignment results can help achieve a significant reduction of the local training time of federated learning through evaluation.

The rest of the paper is organized as follows. In Sect. 3 and Sect. 4, we present the problem definition and system model. In Sect. 5, we present the proposed algorithm in detail. In Sect. 6, we present the experimental evaluation and insights obtained. Section 7 summarizes the paper and outlines future work.

[2] We denote an assignment as a possible connection of the IoT devices to the edge servers in the cluster in a specific fashion.

2 Related Work

In this section, we review the previous works which are related to minimizing the retrieval delay in edge computing and federated learning scenarios.

Yousefpour et al. [51] proposed a policy for minimizing service latency in fog computing networks. Sun et al. [42] proposed a network architecture based on Cloudlet, which minimizes the average response time of mobile users by offloading the workloads. Kherraf et al. [15] proposed workload assignment of IoT devices to the edge server using Tabu search (TS). Nijimbere et al. [32] show that TS will get stuck at a local optimum. Perikan et al. [34] presented an approach for the assignment of an IoT device to the edge server without any discussion about minimizing communication latency issues between IoT and the edge server. Qiang et al. [10] proposed an algorithm called Application Aware Workload Allocation ($AREA$) for assigning the IoT users' requests to the cloudlet to reduce the response time from the cloudlet. Song et al. [42] presented an approach for the periodical distribution of incoming tasks at an edge computing network and satisfying the Quality of Service (QoS) requirements like completion time. However, they did not focus on real-time edge computing applications where the response time is crucial.

Numerous studies on workload offloading in a MEC infrastructure have emphasized end-to-end latency. Sun and Ansari [45] examined a network in which numerous users outsource their tasks to geographically dispersed MEC nodes. They provided a system for latency-aware offloading that lowers the overall response time caused by the workload of the users. In [47], for instance, the authors evaluated the trade-off between energy usage and latency by solving the IoT workload offloading problem. Their suggested approach improves system utility and energy usage while meeting latency requirements.

By jointly optimizing offloading decisions and resource allocation, Sheng et al. [39] suggested a computation offloading system to reduce the average offloading latency of users. Lin et al. [20] presented a double offloading approach in which neighboring edges shared their resources and jointly offloaded user queries in order to more efficiently utilize their communication and computing resources. Nevertheless, this research predominantly employed custom-built heuristic techniques that cannot be generalized. Recent research has demonstrated that deep reinforcement learning has considerable potential for handling complicated decision-making challenges in edge/cloud computing systems. Hua et al. [11] surveyed machine learning integration with edge computing. In this survey, they also reviewed the deep reinforcement learning solutions for optimizing edge computing scenarios with respect to reducing latency. Huang et al. [12] proposed a deep reinforcement learning-based offloading algorithm to evaluate whether tasks for a set of mobile devices should be locally processed or offloaded, and how to allocate wireless bandwidth to optimal performance. Lie et al. [22] proposed a deep reinforcement learning solution for edge server selection. In this selection of edge servers, they consider the aspect of computation offloading such as which edge server to be selected in order to perform computation. However,

they do not consider the edge server selection based on the delay incurred in fetching the data from IoT devices.

Nonetheless, these investigations consider minimizing the workload delay and focusing on reliability without considering the delay incurred in fetching the data from IoT devices. None of the mentioned work talks about the assignment of IoT devices to the edge servers based on the fetching delay incurred. In our work, we focused on minimizing overall retrieval delay incurred from the IoT devices to the edge servers.

Minimizing Data Retrieval Time for Federated Learning: The rate at which data reaches local devices, in our case edge servers, can have an effect on the training time of federated learning. Edge servers train the model using their own local data and then communicate the updated model parameters to a centralized server for aggregation [17,19].

In numerous ways, the training performance of federated learning can be improved if the data comes fast to the local device. Firstly, it can shorten the time required to train the model locally, enabling the device to contribute more frequently to the training process. Secondly, it can enhance the quality of local training since the model can learn from more recent data that more accurately reflects the current state of the system [13,17]. Overall, minimizing retrieval data is a crucial performance aspect of federated learning [19].

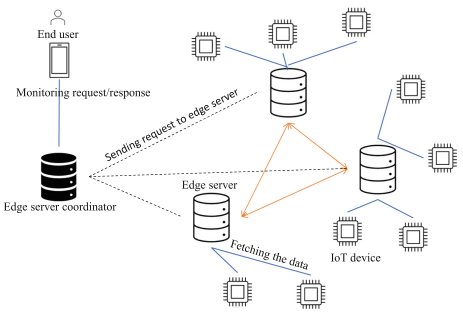


Fig. 1. Edge computing environment

3 Problem Statement: The *EAP* is NP-Hard

In this section, we formally define *the Edge Assignment Problem* (EAP) which we mentioned already in Sect. 1.

Let there be m edge servers and n IoT devices in the network. We denote $I = \{1, 2, \ldots, m\}$ and $J = \{1, 2, \ldots, n\}$; also, $i \in I$ and $j \in J$ denote the indices of an edge server and an IoT device. Let:

– p_{ij}, r_{ij} denote the *retrieval delay* and the *load* on the ith edge server, when the jth IoT device is connected to the ith edge server.

- b_i denote the *maximum load* the ith edge server can sustain.
- x_{ij} be a binary variable to denote if the jth IoT device is connected to the ith edge server.

Now the problem is: given $\{p_{ij}\}$, $\{r_{ij}\}$ and $\{b_i\}$, determine:

$$\min\{\max_{i\in I, j\in J}\{p_{ij}\cdot x_{ij}\}\} \tag{1}$$

subject to:

$$\sum_{j\in J} r_{ij}\cdot x_{ij} \leq b_i, \forall i \in I, \tag{2}$$

$$\sum_{i\in I} x_{ij} = 1, \forall j \in J, \tag{3}$$

$$x_{ij} \in \{0,1\}. \tag{4}$$

Constraint Eq. (2) says the sum of the loads on each edge server should not exceed its maximum capacity. Constraint Eq. (3) says each IoT device must be connected to exactly one edge server.

The EAP as described above is a well-studied problem under the name of the BGAP; the BGAP stands for the *Bottleneck (version of the) Generalized Assignment Problem (GAP)*. Both the GAP and the BGAP belong in the complexity class NP-Hard [25,26,28]. They are NP-Hard in the strong sense, which means that – even if we ignore the objective function (i.e., (1)) in both the problems – it is even hard to determine the existence of an assignment for $\{x_{ij}\}$ that may satisfy (2). For the sake of completeness, we reproduce the proof below.

Theorem 1. *The EAP is NP-Hard.*

Proof. We reduce the NP-Complete 2-set-partition problem to the EAP. The input to the algorithm that solves 2-set-partition is a set S containing n non-negative real numbers: $\{A_1, A_2, ..., A_n\}$; the output is 1, iff there exists a subset S_1, such that $sum(S_1) = sum(S \setminus S_1)$. In the EAP we set $m = 2$, $A_j = r_{1j} = r_{2j}, \forall j \in J$ and $b_1 = b_2 = \frac{\sum_{j\in J} A_j}{2}$. While mapping the instances of 2-set-partition to those of the EAP, we ignore b_1, b_2 and $\{p_{ij}\}$. Suppose, the algorithm $\mathcal{A}$ solves the EAP; it does so by outputting the optimal value, or by outputting $\perp$ denoting that no solution exists. It is easy to see that $\mathcal{A}$ also solves the 2-set-partition as follows:

1. If it outputs $A \neq \perp$ then set partition exists.
2. If it outputs $A = \perp$ then the set partition does not exist.

The above proof also establishes that even if we relax the EAP with $r_{1j=r2j}$ (call this problem EAP′), then EAP′ is also NP-Hard.

Let us concentrate on the following special cases of EAP′: $r_{ij} = 1$, $b_i = \mathcal{T}$, and $n = \mathcal{T}m, \forall i, j$. Interestingly, when $\mathcal{T} = 1$ (call this problem EAP″) then it is poly-time solvable [26]; but when $\mathcal{T} > 1$ (call this problem EAP‴), the

problem seems hard (although the proof is missing). In our case: we assumed that $\mathcal{T} > 1$.[3]

In summary, 2-set-partition $\leq_P$ EAP$'$ $\leq_P$ EAP; EAP$''$ $\leq_P$ EAP$'''$ $\leq_P$ EAP$'$ $\leq_P$ EAP. The EAP$''$ is poly-time solvable; the EAP$'$ and EAP are both NP-Hard; but the status of EAP$'''$ is not yet known. We have done extensive analysis of EAP$'''$, and have conjectured that it is also NP-Hard. Due to space constraints, here we omit the full discussion which will appear in the extended version.[4]. In the remainder of the paper, we solve EAP$'''$ using heuristics and machine learning techniques.

4 System Model

We consider an intelligent edge computing environment comprising IoT devices and edge servers as shown in Fig. 1. We assume that the IoT devices are equipped to do the following jobs: filtering, preprocessing, and transmitting data to the edge server for analysis purposes [29]. Edge servers host containerized applications or services intended to process the data from the respective IoT devices [2]. Each IoT device is connected to an edge server via a wired or wireless medium.

An end user submits a service request to the edge cluster. The request is handled by the *coordinator*, i.e., a designated edge server, which subsequently schedules the request to one or more edge servers. The above edge servers then proceed to retrieve the required data from one or more IoT devices. We consider the effect of the above retrieval delay, defined earlier, often referred to as the *communication time* [18], which has a significant effect on the overall turnaround time of the service [6]. Retrieval delay can be determined in a practical IoT deployment by employing a network monitoring tool in the edge server.[5]

We give special consideration to federated learning, which is a special case of edge computing and distributed machine learning, in which each edge server (i.e., local client) trains a local model using training data collected from IoT devices located close to said edge server. The observed local training time, i.e., the end-to-end delay in learning the local models, is directly proportional to the retrieval delay, i.e., the delay experienced by the edge servers in collecting the training data from the IoT devices assigned to it. Hence, an optimal assignment would minimize the above retrieval delay, which in turn, would result in a reduction of the overall local training time. In this paper, we work with the following assumptions.

- A-1: All edge servers and IoT devices are stationary, i.e., they rarely change their geographical locations. This assumption is justified by a host of edge

[3] For example, $\mathcal{T} = 2$ means that the maximum number of IoT devices that can be assigned to an edge server is 2.

[4] The conjecture is based on the fact that $\binom{\mathcal{T}n}{m}$ is constant when $\mathcal{T} = 1$, but exponential when $\mathcal{T} > 1$.

[5] https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html.

computing applications, such as oil refinery monitoring [38], traffic monitoring [5] etc., where edge servers are stationary in nature.[6]

– A-2: We consider a threshold to the number of IoT devices that can be connected to an edge server. We assume that this threshold is identical for all the edge servers. The edge servers perform certain analytics workloads on the data retrieved from the IoT devices, and in doing so they require some computing and storage resources. Such resources are typically limited within a pre-defined threshold for edge servers. Hence, the above resource limitations, i.e., the thresholds provided by the hardware manufacturers or the end users themselves, must be considered as an additional constraint in the given optimization problem of assigning the IoT devices to the edge servers [6].

5 Reinforcement Learning (RL) for EAP

In this section, we elaborate on the reasoning behind our decision to use *RL*-based algorithms to solve the given *EAP*. Though in the prior literature, traditional heuristic algorithms have been proposed to solve the *assignment problem*, which is similar to Eq. (1), most of these algorithms have been unable to show improvements beyond local optima [16,27,32]. Further, according to *No Free Lunch* (*NFL*) theorem, a single heuristic is not suitable for solving multiple optimization problems [49]. To that end, many researchers have proposed machine learning-based approaches to solve different variations of the assignment problem [21,33]. However, classical machine learning techniques are generally used to solve a problem where it is safe to assume that data are independent and identically distributed (*i.i.d*). On the other hand, in our case, prior information about the assignments of the IoT devices to the edge servers is not readily available. Hence, there is a lack of availability of training data to build a classical machine learning model. Because of the above reason coupled with the absence of any prior information about the distribution of the data, the classical machine learning algorithms are not suitable for our problem. Hence according to [46], the defined problem requires a trial and error-based approach to learning. Specifically, researchers have used Reinforcement Learning (*RL*) algorithms in the past to solve complex optimization problems without relying on training data [4,22]. Following [16,27,32], we express the *EAP* as a *Reinforcement Learning* (*RL*) problem.

5.1 The MDP Model for EAP

Here, we model *EAP* as *Markov Decision Process (MDP)* as follows.

– **State space** (S)**:** A state $s \in S$ is an n-dimensional vector, $s = [s_1, s_2, \ldots, s_n]$ where $\forall j \in J$, $s_j \in I$, represents the edge server which the jth IoT device is assigned to. If $s_j = 0$, the jth IoT device has not yet been

[6] In future work, we consider the mobility of IoT devices and edge servers which makes the problem more challenging.

assigned to any edge server. If $s_j = i$, where $i \in I$, the jth IoT device has been assigned to the ith edge server. We define s^T as a *terminal state* where all the IoT devices have been assigned, i.e., $\forall j \in J, s_j \neq 0$
For example, let us consider a problem instance with n = 4 IoT devices and m = 2 edge servers. A possible state vector s could be: $s = [0, 1, 0, 2]$. In this example, the first and third IoT devices have not yet been assigned to any edge server ($s_1 = 0$ and $s_3 = 0$). The second IoT device has been assigned to the first edge server ($s_2 = 1$), and the fourth IoT device has been assigned to the second edge server ($s_4 = 2$).

- **Action space (A):** An action $a \in A$ corresponds to assigning an IoT device to an edge server. Action space of the system A is defined as $A = \{a | a \in \{x_{11}, x_{12}, x_{13} \cdots, x_{nm}\}\}$. Generally, an agent interacts with the environment and takes appropriate action. Here, an action space of the system is the set of all *possible* assignments of IoT devices that an agent is going to take on the basis of the observed state. In each state, the agent can choose from a subset of actions in A, considering only the unassigned IoT devices and ensuring the threshold of the edge server given in Eq. (2).
  For example, let us consider a problem instance with n = 3 IoT devices and m = 2 edge servers. The complete action space A can be defined as $A = \{a | a \in \{x_{11}, x_{12}, x_{21}, x_{22}, x31, x_{32}\}\}$. For instance, if IoT device 1 has already been assigned to an edge server and the threshold of edge server 2 is reached, then the available actions for the RL agent in this state would be a subset of A, i.e., $a \in \{x_{21}, x_{31}\}$. The action space at the terminal state s^T is null because no further actions are required as all the IoT devices have been assigned to the edge servers.
- **Transition probabilities** P**:** The transition probability function $P(s'|s, a)$ defines the probability of transitioning from state s to state s' when action a is performed. In this problem, transitions between states are deterministic. If action a is valid i.e., assigning the IoT device to the edge server doesn't violate the threshold constraint, then the probability of transitioning to state s' is 1, and 0 otherwise.
- **Reward (R):** A reward is a scalar indication of an agent's performance towards achieving the goal. The goal here is to minimize the maximum retrieval delay such that none of the edge servers is overloaded as per (1). In our MDP, the reward function R(s, a, s') returns the immediate reward for transitioning from state s to state s' by taking action a which is defined as follows
  - $R(s, a, s') = -M$, where M arbitrarily large constant. If the action is invalid, i.e., the IoT device j has already been assigned, or the capacity constraint of the edge server i is violated, the reward function returns a large negative reward $-M$ to discourage the agent from taking invalid actions.
  - If the action is valid and all IoT devices have been assigned to edge servers after taking the action, the reward function returns the negative maximum retrieval delay in the new state s'. In this case, the reward function is defined as $R(s, a, s') = -max\_delay(s')$, where $max\_delay(s')$

is the maximum retrieval delay observed among all IoT devices for a particular assignment in state s'.

- $R(s, a, s') = 0$, If the action is valid, but not all IoT devices have been assigned to edge servers after taking the action.

The reason for choosing the above reward function follows from our original goal to minimize the maximum retrieval delay incurred in retrieving the data from the IoT devices to the edge servers. At the same time, the *RL* agent is discouraged to perform invalid actions.

5.2 Preliminaries

Here, we introduce the various concepts and terminology we have used in RL algorithms.

- **Policy (π):** A policy is a function that maps the state space to the action space. In our case, the policy is deterministic and represented as a function $\pi : S \rightarrow A$. In the case of EAP, the deterministic policy will provide the best action, i.e., the particular assignment to perform in each state on the basis of the current assignment and the threshold on each edge server. The aim of agent learning is to determine an optimal edge assignment defined earlier. An agent uses the policy to learn the optimal actions to reach a solution to the given problem. A policy should consider the exploration and exploitation trade-off in training the agent [46]. For the given problem, we consider the ϵ-*greedy* policy since it ensures that there is a healthy balance of exploitation[7] and exploration of the state-action pairs by the agent. We can configure the ϵ-*greedy* policy to perform *exploration* and *exploitation* with a probability $\frac{\epsilon}{|A(S)|}$ and $1 - \epsilon + \frac{\epsilon}{|A(S)|}$, respectively.
- **Action-Value (Q):** In RL, we use a value function to define the expected future rewards in terms of the current state or state-action pair. Since, we do not know the full dynamics of the environment, i.e., the probability of the action taken in a given state, dynamic programming solutions cannot be applied. Therefore, action-value function (Q) is used instead of the state-value function (V) [3,46]. As per the standard practice in RL [41], the action-value function $Q_\pi(s, a)$ is expressed in terms of the following Bellman equation.

$$Q_\pi(s, a) = Q_\pi(s, a) + \alpha \left[R_{t+1} + \gamma Q_\pi(s', a') - Q_\pi(s, a)\right] \tag{5}$$

where π denotes the policy, $a \in A$ denotes the action, $s \in S$ denotes the state, and γ denotes the discount factor, which quantifies the relative importance assigned to the future action rewards with respect to the current action reward. Therefore, Eq. (5) estimates the action-value based on the immediate reward obtained based on action a, and discounts the estimated action value

[7] Here exploitation means acting greedily with respect to the action-value function [31].

by *discount factor* (γ). If we use max on discounted action-value, action-value $Q(s,a)$ is defined as

$$Q_\pi(s,a) = Q_\pi(s,a) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q_\pi(s',a') - Q_\pi(s,a) \right] \tag{6}$$

In Eq. 6, the expression $R_{t+1} + \gamma \max_{a'} Q_\pi(s',a')$ is called as *temporal difference target*. The optimal policy is defined as

$$\pi^*(a|s) = \arg\max_{a \in A} Q^*(s,a) \tag{7}$$

where $Q^*(s,a)$ is an optimal action value. This optimal policy $\pi^*(a|s)$ gives an action $a \in A$ where the estimated action-value $Q^(s,a)$ is maximum. As the agent does not know about the environment, we make use of the time difference of learning without any prior information about the environment [46]. The value of the state (V) under an optimal policy is expressed as *Bellman optimality equation* as follows.

$$v_*(s) = \max_{a \in A} Q_{\pi*}(s,a) \tag{8}$$

– **Episode:** Owing to the exploratory nature of RL algorithms, the agent continues the learning in the form of a series of episodes - each of which represents a particular sequence of tuples $(state, action, reward)$ generated on the basis of the policy followed. This sequence will continue till the terminal state s^T.

As elaborated with examples in Sect. 5.1, the state space in our case is a collection of state vectors, where each state vector is given as a finite set of integers. Similarly, the action space is given as a set of binary variables x_{ij} where each $x_{ij} \in \{0,1\}$. Hence, both state space and action space are discrete in nature[8]. We consider real-world large-scale edge deployments, where the number of edge servers and IoT devices is typically very large. On the other hand, the popularly used reward-based learning algorithms such as tabular methods are able to produce an optimal solution only for smaller discrete state space and action space [46]. Among reward-based learning methods, *Temporal Difference* (TD) learning is widely used in practice [8,40]. Popular tabular TD algorithms, such as $SARSA$ and *Q-learning*, have been frequently applied by researchers in the systems and networking domain [24]. However, with an increasing number of IoT devices and edge servers commonly encountered in our case, state and action space becomes too large to handle according to the above techniques.

As per [30], the above issue is tackled commonly by applying function approximation, and among the existing function approximation techniques, researchers have commonly used *neural networks*. Particularly for very large and discrete state space and action space, Deep Q network (DQN) is a popular choice to approximate the value function for each possible action of a state. In the next subsection, we elaborate on our DQN-based RL algorithm for EAP.

[8] In future work, we consider continuous state and action spaces since the edge servers and IoT devices are going to be non-stationary.

5.3 Deep Reinforcement Learning (DQN) for EAP

In DQN, the neural network receives the current state as input and generates a Q-value for each possible action. The action with the highest Q-value is selected, and the agent interacts with the environment by performing that action and observing the subsequent reward and state. The following Q function is used by the agent to update the Q-values.

$$Q(s, a, \theta) \approx Q^*(s, a), \tag{9}$$

where $Q(s, a, \theta)$ denotes the Q value of taking the action in a given state s, and is estimated with respect to the weights of the neural network θ. As a result of the above interactions with the environment, the agent learns from experience by iteratively changing its Q-function based on the reward assigned as per Subsect. 5.1.

Nevertheless, in most RL algorithms, the training data samples are frequently strongly correlated, resulting in instability and unsatisfactory performance [46]. To tackle the issue of instability, a *replay buffer* is typically used to store the agent's past experiences and perform random sampling randomly on it during training. Each experience is saved as a tuple (s, a, r, s') comprising the current state s, the action performed a, the resultant reward r, and the future state s'. As an agent interacts with its surroundings, these tuples are added to the replay buffer. During training, a random sample of events from the replay buffer is used to update the Q-function. By sampling at random from the replay buffer, the training data become less correlated and more representative of the actual distribution of events.

Deep Q-learning uses two networks to stabilize the training process from overestimation bias of action-value values, i.e., the policy network and the target network. Throughout the training process, the policy network is used to determine actions, and its weights are adjusted with each iteration to increase its performance, such as training stability. In contrast, the target network is used to evaluate the Q-values of the policy network, and its weights θ_t are fixed for a specified number of iterations before the agent starts to adjust θ_t to match it with the weights of the policy network θ. In the absence of the target network, the policy network would be modified depending on the estimated Q-values at each iteration, resulting in a feedback loop that can lead to overestimation bias and instability in the training process. During the training, the following standard loss function $L(\theta)$ is being used.

$$L(\theta) = E[(r + \gamma max_{a'} Q(s', a'; \theta_t) - Q(s, a; \theta))^2] \tag{10}$$

The DQN algorithm for agent training is shown in Algorithm 1. In lines 2–4, we initialize replay memory D with maximum capacity C. Replay memory stores the transitions (s, a, r, s') that the agent observes during its interactions with the environment. It is used to sample batches of transitions to train the Q-network - a neural network with random weights θ. The Q-network accepts as input a state s and outputs a Q-value for each possible action a. The action a taken in state s

Algorithm 1. Deep Q-Learning for EAP

Input: $\{p_{i,j}\}, n$ and $\mathcal{T}$ ▷ Inputs to the algorithm
Initialization : ▷ Initializing Reply buffer, Q network, Target network, and hyper parameters
Initialize Replay Buffer D with maximum capacity C
Initialize Q-network with random weights θ, target network with $\theta_t = \theta$
Initialize Parameters : exploration rate ϵ, exploration decay ϵ_{decay}, discount factor γ, learning rate α, number of episodes E, update frequency of target network U
Procedure:
for episode 1 to E **do**
    reset the state ▷ Here all the IoT devices are unassigned with edge servers
    set the episode reward $\mathcal{R} = 0$
    while episode not done **do**
        With probability ϵ select random action a ▷ assign IoT to any available edge server randomly
        Otherwise select action $a = \arg\max_a Q(s, a; \theta)$ ▷ assign IoT device to an edge server which results in maximum action value estimate
        Take action a, and observe R(s, a, s') and s'
        Store transition $(s, a, R(s, a, s'), s')$ in replay memory D
        Sample minibatch of transitions from D
        if episode not done **then**
            $Q_{target} = R(s, a) + \gamma * max'_a Q(s', a'; \theta_t)$
        else:
            $Q_{target} = R(s, a)$
        end if
        Update Q-network weights θ by minimizing loss: $L(\theta) = (Q_{target} - Q(s, a; \theta))^2$
        Every U steps, update $\theta_t = \theta$
        $\epsilon = \epsilon * \epsilon_{decay}$
        s = s'
        R = R + r
    end while
end for

is based on the estimated Q-value, and this estimation is performed by a second neural network called the target Q-network since it is used to generate the target Q-values for training the Q-network. The weights of the target Q-network are updated less frequently than the Q-network's weights to make the training more stable.

Line 5 sets the hyperparameters used for training the DQN agent. The discount factor γ determines the importance of future rewards. The learning rate α controls how much the weights of the Q-network are updated in response to the training data. The exploration probability ϵ determines the probability that the agent selects a random action instead of the action with the highest Q-value. The exploration decay rate ϵ_{decay} determines how quickly the exploration prob-

ability decreases over time. The update frequency U determines how often the weights of the target Q-network are updated.

Lines 6–22 explain the steps undertaken in training the DQN agent. For each episode, we reset the state of an environment and set the reward of the episode to 0. For each step in an episode, we select a random action a based on the exploration probability ϵ. Otherwise, the action with the highest Q-value is selected. Here, each action corresponds to a particular assignment of an IoT device to a given edge server. The agent performs the action a and observes a reward r and the next state s'. The transition tuple (s, a, r, s') is stored in replay memory. Now, we sample a minibatch of transitions from the replay buffer to train the Q-network, and subsequently, the target Q values for training the Q-network are computed. If an episode is unfinished, the target Q-value is expressed in terms of the immediate reward plus the discounted maximum Q-value, otherwise, the immediate reward itself is assigned to the target Q-value. Next, we update the Q-network weights θ while minimizing the loss given in 10. In lines 22–25, for every step U, we update the weights of the target network θ_t with the weights of the Q network θ, followed by an update of the exploration probability, the state, and the reward.

6 Experimental Evaluation

6.1 Experimental Setup

We run the simulation on a six-core processor (3.6 GHz), with a memory of 32 GB and an Ubuntu 20.04 LTS. To demonstrate the performance of our proposed *RL* algorithms, we created a custom environment using Gymnasium[9] - Python toolkit widely used for this purpose. For realistic analysis, the values of retrieval delays used in the experiments are estimated based on measurements done on public network platforms such as the PlanetLab data set [52] which comprises of estimates for the end-to-end delays between devices, usually falling in a range of 3 to 352 ms.[10] We randomly assign these estimated delays to our experimental cluster configurations comprising IoT devices and edge servers. This is because we randomly assign the retrieval delays during the training process so that the trained model would adapt to the variances in the retrieval delays incurred. We run our experiments for 500 to 1500 episodes, while the rest of the parameters are defined in Table 1.

6.2 Evaluation Methodology

Following the standard practice in prior papers in this area [7,22], we run simulation experiments of our proposed algorithms by varying the following hyperparameters. The first hyperparameter is the *configuration* of a cluster which denotes

[9] https://gymnasium.farama.org/.

[10] In most near real-time applications, the upper latency limit is 300 ms. Further, in the PlanetLab dataset used, the latency actually falls within the chosen range of 3–352 ms. We assume that the streaming latency does not vary much with the nature of the data.

Table 1. Table of Parameters

Number of IoT devices	2000
Number of edge servers	100
$\mathcal{T}$	20
Exploration rate (ϵ)	1 Decayed to 0.1
Learning rate (α)	0.5 Decayed to 0.0001
Discount factor (γ)	0.99
replay buffer size	10000
mini batch size	64
update frequency U	50

the number of edge servers and IoT devices, along with the relative positioning of the devices due to their geographical distribution. Due to changes in the geographical distribution of IoT devices and edge servers, retrieval delay between each pair of IoT devices and edge servers may vary. Therefore, we choose random configurations in our experiments, where each configuration has different associated retrieval delays. The other hyperparameter is the number of episodes for the RL algorithm. We record variations in the observed retrieval delay and mean reward obtained with respect to the changes in the above hyperparameters. We evaluate our proposed algorithms against the following baseline algorithms.

- First, we evaluate with respect to the nearest neighbor search (NNS) which is a popular and efficient greedy algorithm used in similar problems in systems and networking [48]. NNS assigns each IoT device to an edge server based on a *greedy property*, where an IoT device is assigned to an edge server in a greedy manner such that the retrieval delay is minimized. If an edge server exceeds its threshold $\mathcal{T}$ with the above assignment, then the concerned IoT devices are assigned to the next available edge server. The greedy property of NNS fails to find the global optimal solution since it considers only the locally optimal assignments.
- Next, we consider the *Tabu Search* (TS) - meta-heuristic optimization technique for search-based problems which was used by Kherraf et al. for assigning the IoT workloads to the edge servers while reducing the completion time [15]. While evaluating TS, we initialize the TS instance with the solution obtained by applying the NNS baseline described above. TS stores metadata of *tabu moves*[11] and the recent list of solutions found in the search procedure. We consider a tabu move as an assignment that results in a maximum retrieval delay. While maintaining the solution list, we avoid considering the tabu moves up to certain iterations to find an improvement in the currently obtained solution. In each iteration, TS will try to improve the solution by swapping the

[11] A tabu move is a forbidden move that cannot be included in the recent solution list obtained.

tabu move with an assignment that results in a minimum retrieval delay The search procedure halts if the specified iterations have finished, or if the solution gets stuck at one point in the search space. The drawback of this baseline is that TS will get stuck at some point, i.e., further tabu moves cannot be swapped since the metadata contains partial knowledge of the assignments performed [32].

– Next, we evaluate our proposed algorithm against Google OR-Tools solver [35] - a popular optimization problem solver. Specifically, we run the SCIP solver[12] package - a mixed integer programming solver from Google OR-Tools. For that purpose, we transform the CAP$'''$ to a minimization problem as follows [1].

$$\begin{aligned} \min \quad & \mathcal{Z} \\ \text{subject to} \quad & \mathcal{Z} \geq p_{i,j} x_{i,j}, i = 1, \ldots, n, j = 1, \ldots, m, \\ & \sum_{i \in I} x_{i,j} = 1 \; j \in J \\ & \lfloor \frac{n}{m} \rfloor \leq \mathcal{T}. \end{aligned}$$

With the above experiment we get the optimal solution.

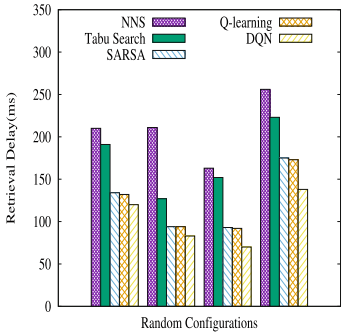
– As a last baseline, we compare with *Q-learning* and *SARSA*.

6.3 Experimental Results

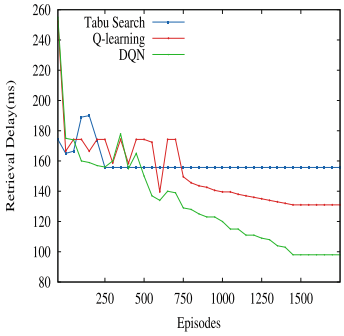
In Fig. 2(a), we illustrate the retrieval delay for different random configurations. We observe that the NNS approach results in a higher observed retrieval delay than TS, whereas tabular RL algorithms *Q-learning* and $SARSA$ perform betters than TS and NNS. In contrast, DQN is able to reduce the overall retrieval delay much more than all the baselines. Figure 2(b) shows that during the initial episodes, DQN results in a higher retrieval delay than TS. This is because TS employs a greedy NNS-based technique as its initial solution. In contrast, *Q-learning* first performs random assignment. Hence, the assigned IoT devices are not always nearest to the edge servers causing an increased retrieval delay. While the number of episodes increases, TS may become fixated on a single solution. On the other hand, *Q-learning* maintain a trade-off between exploration and exploitation - exploring, i.e., assigning an IoT device to an edge server that it has never seen before, and exploiting, i.e., assigning an IoT device to an edge server that it has seen before. It keeps learning various possible assignments from the environment and then exploits based on these discoveries. Note that one of the configurations from Fig. 2(a) was used for the rest of the experiments. The trends shown in Fig. 2(c) to Fig. 2(f) are consistent for all the configurations.

In Fig. 2(c), we evaluate $SARSA$, *Q-learning*, and DQN based on the mean reward obtained. During the exploration process of RL algorithms, mean reward

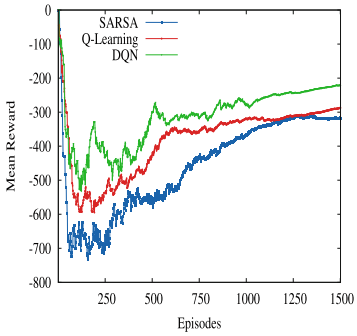
[12] https://www.scipopt.org/.



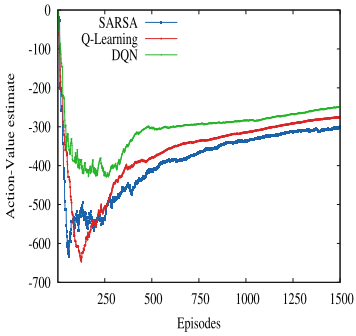
(a) Retrieval delay observed with different assignments of IoT devices to edge servers.



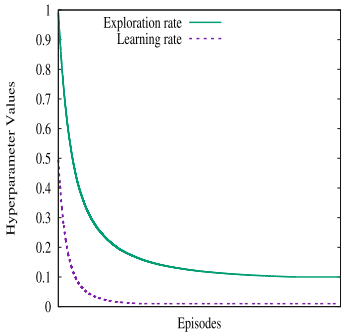
(b) Retrieval Delay observed with increase in the number of episodes where TS fixates on a one sub optimal solution whereas DQN provides much-reduced retrieval delay.



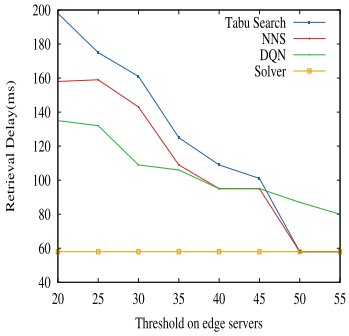
(c) Mean reward obtained during different episodes.



(d) Value Estimates in different episodes.



(e) Epsilon and Alpha Decayed values different epsiodes.



(f) Impact of increasing $\mathcal{T}$ at the edge servers on retrieval delay.

Fig. 2. Experimental Results with observed retrieval delay and mean reward.

includes the delayed rewards[13], i.e., a current action taken by the agent must take into account the immediate and future rewards. In Fig. 2(c), 2(d), we plot the mean reward obtained and estimated value of a state with respect to an increasing number of episodes. The mean reward obtained and value estimate will fluctuate in the initial episodes. Later, we observe that DQN gets a higher mean reward and value estimate than *Q-learning* and $SARSA$. Therefore, DQN shows better convergence to an optimal policy than tabular RL algorithms.

In Fig. 2(e), we demonstrate the decay of the exploration rate with the increase of the decay ratio. Following the standard practice [31], we set the exploration rate initially to 1, and the decay ratio degraded from 0.9 to 0.1. The initial higher values of exploration rate result in a higher degree of exploration and lesser exploitation. The learning rate also decayed from 0.5 to 0.01.

Table 2. Comparison of retrieval delays with varying sizes of IoT devices and edge servers

IoT devices	edge servers	OR-Tools Solver		Deep *Q-learning*	
		Iterations	Retrieval Delay (ms)	Episodes	Retrieval Delay (ms)
1000	50	1025	129	500	136.8
1500	100	2045	165.4	500	174.5
2000	100	3195	175.4	1000	162.5
2500	200	5195	198.40	1500	172.5

Following [33], we conduct similar experiments on Google or-tools with an increasing number of IoT devices and edge servers *solver*. We compared it with the DQN algorithm and recorded the observed retrieval delay in Table 2. We limited the solver iterations according to the episodes performed in DQN. Also, we conducted this experiment at approximately the same wall clock time for the solver and DQN. For smaller instances, the solver performs better than DQN. As the number of IoT devices and edge servers increases, DQN outperforms the solver solution. Hence, the RL approach scales better than the solver with an amortized cost of training.

Figure 2(f) illustrates the impact of increasing $\mathcal{T}$ at the edge servers on the observed retrieval delay for NNS, TS, solver, and DQN. Initially, DQN performs better than the baseline approaches. After increasing the $\mathcal{T}$ to a certain extent, NNS and TS match the solution obtained with the solver, whereas DQN gives a higher retrieval delay. This is because of unexplored assignments of the IoT devices to the edge servers in contrast to the greedy approach followed by NNS and TS. Here, we observe that the DQN is not converging with the solver solution whereas NNS and TS converged on the solver solution [6]. Though we increased $\mathcal{T}$ in our experimental setting, such variations are actually impractical in real scenarios since the edge server are typically resource constrained and are

[13] Delayed reward is a reward obtained after multiple episodes.

able to process data from multiple sources up to a certain threshold only. This is because edge servers are resource constrained and able to process a limited number of IoT devices only [6].

6.4 Assisting Federated Learning

As elaborated in Sect. 1, our proposed algorithm EAP is useful in the case of federated learning (FL), particularly in reducing the training time incurred at the edge servers (FL clients) in learning local models. Here, the edge servers and the IoT devices act as clients and data sources, respectively. The retrieval delay, i.e., the time in fetching the training data from IoT devices, has a direct impact on the local training time at the edge servers. Hence, in reducing the retrieval delay, our proposed algorithm is actually able to minimize the local training time [17]. For this experiment, we use cifar100 data set[14] and train an image classification model on different edge servers in a federate learning setting. Figure 3 illustrates that our approach results in considerably reducing the local training time compared to the baseline algorithms. Also, the figure illustrates that the retrieval delay consists of a large fraction of the local training time. Hence, reducing the retrieval delay is quite important in the case of FL. Note that we have considered that local models are trained only after the local training data has been retrieved. In federated learning systems, relatively large retrieval delays can result in excessively outdated local models in comparison with the global model. As a result, it would take the model aggregation at the client to take a much larger time in achieving convergence [17]. Thus, our approach is able to reduce the overall delay in federated learning.

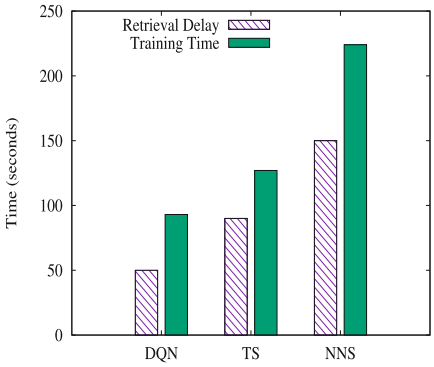


Fig. 3. Reduced local training time in federated learning.

[14] https://www.cs.toronto.edu/~kriz/cifar.html.

7 Conclusion and Future Work

In this work, we studied an intelligent assignment of the IoT devices to the edge servers to minimise the retrieval delay while ensuring that none of the edge servers is overloaded. We prove the problem is NP-hard even when the IoT devices and edge servers are stationary. To that end, we modeled the problem as an MDP framework and leveraged neural network-based Deep Q-learning to determine an optimal assignment of the IoT devices to the edge servers in a given cluster. In future work, we plan to consider IoT devices and edge servers to be non-stationary, which increases the problem's difficulty. Further, we would perform a much more in-depth analysis of the impact of EAP on different performance aspects of federated learning.

References

1. Design optimization. http://apmonitor.com/me575/index.php/Main/MiniMax
2. Whitepapers. https://docs.aws.amazon.com/whitepapers/latest/develop-deploy-dotnet-apps-on-aws/running-applications-in-containers.html
3. Abramson, M., Wechsler, H.: Tabu search exploration for on-policy reinforcement learning. In: 2003 Proceedings of the International Joint Conference on Neural Networks, vol. 4, pp. 2910–2915. IEEE (2003)
4. Ahsan, W., Yi, W., Liu, Y., Qin, Z., Nallanathan, A.: Reinforcement learning for user clustering in NOMA-enabled uplink IoT. In: 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6. IEEE (2020)
5. Barthélemy, J., Verstaevel, N., Forehead, H., Perez, P.: Edge-computing video analytics for real-time traffic monitoring in a smart city. Sensors **19**(9) (2019). https://doi.org/10.3390/s19092048, https://www.mdpi.com/1424-8220/19/9/2048
6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2342509.2342513
7. Brockman, G., et al.: OpenAI gym. arXiv preprint arXiv:1606.01540 (2016)
8. Chien, W.C., Weng, H.Y., Lai, C.F.: Q-learning based collaborative cache allocation in mobile edge computing. Futur. Gener. Comput. Syst. **102**, 603–610 (2020)
9. Desikan, K.S., Kotagi, V.J., Murthy, C.S.R.: Topology control in fog computing enabled IoT networks for smart cities. Comput. Netw. **176**, 107270 (2020)
10. Fan, Q., Ansari, N.: Application aware workload allocation for edge computing-based IoT. IEEE Internet Things J. **5**(3), 2146–2153 (2018)
11. Hua, H., Li, Y., Wang, T., Dong, N., Li, W., Cao, J.: Edge computing with artificial intelligence: a machine learning perspective. ACM Comput. Surv. **55**(9), 1–35 (2023)
12. Huang, L., Bi, S., Zhang, Y.J.A.: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**(11), 2581–2593 (2020). https://doi.org/10.1109/TMC.2019.2928811
13. Kairouz, P., et al.: Advances and open problems in federated learning. Found. Trends® Mach. Learn. **14**(1–2), 1–210 (2021)

14. Kherraf, N., Alameddine, H.A., Sharafeddine, S., Assi, C.M., Ghrayeb, A.: Optimized provisioning of edge computing resources with heterogeneous workload in IoT networks. IEEE Trans. Netw. Serv. Manage. **16**(2), 459–474 (2019)
15. Kherraf, N., Sharafeddine, S., Assi, C.M., Ghrayeb, A.: Latency and reliability-aware workload assignment in IoT networks with mobile edge clouds. IEEE Trans. Netw. Serv. Manage. **16**(4), 1435–1449 (2019). https://doi.org/10.1109/TNSM.2019.2946467
16. Khosravanian, R., Mansouri, V., Wood, D.A., Alipour, M.R.: A comparative study of several metaheuristic algorithms for optimizing complex 3-D well-path designs. J. Pet. Explor. Prod. Technol. **8**(4), 1487–1503 (2018). https://doi.org/10.1007/s13202-018-0447-2
17. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
18. Lai, F., You, J., Zhu, X., Madhyastha, H.V., Chowdhury, M.: Sol: fast distributed computation over slow networks. In: NSDI, vol. 20, pp. 273–288 (2020)
19. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: challenges, methods, and future directions. IEEE Signal Process. Mag. **37**(3), 50–60 (2020). https://doi.org/10.1109/MSP.2020.2975749
20. Lin, K.C.J., Wang, H.C., Lai, Y.C., Lin, Y.D.: Communication and computation offloading for multi-rat mobile edge computing. IEEE Wirel. Commun. **26**(6), 180–186 (2019). https://doi.org/10.1109/MWC.001.1800603
21. Liu, D., Kong, H., Luo, X., Liu, W., Subramaniam, R.: Bringing AI to edge: from deep learning's perspective. Neurocomputing (2021)
22. Liu, H., Cao, G.: Deep reinforcement learning-based server selection for mobile edge computing. IEEE Trans. Veh. Technol. **70**(12), 13351–13363 (2021). https://doi.org/10.1109/TVT.2021.3124127
23. Liu, S., Liu, L., Tang, J., Yu, B., Wang, Y., Shi, W.: Edge computing for autonomous driving: opportunities and challenges. Proc. IEEE **107**(8), 1697–1716 (2019). https://doi.org/10.1109/JPROC.2019.2915983
24. Liu, X., Yu, J., Wang, J., Gao, Y.: Resource allocation with edge computing in IoT networks via machine learning. IEEE Internet Things J. **7**(4), 3415–3426 (2020). https://doi.org/10.1109/JIOT.2020.2970110
25. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, Hoboken (1990)
26. Martello, S., Toth, P.: The bottleneck generalized assignment problem. Eur. J. Oper. Res. **83**(3), 621–638 (1995)
27. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: a survey. Comput. Oper. Res. **134**, 105400 (2021). https://doi.org/10.1016/j.cor.2021.105400, https://www.sciencedirect.com/science/article/pii/S0305054821001660
28. Mazzola, J., Neebe, A.: Bottleneck generalized assignment problems. Eng. Costs Prod. Econ. **14**(1), 61–65 (1988)
29. Min, M., Xiao, L., Chen, Y., Cheng, P., Wu, D., Zhuang, W.: Learning-based computation offloading for IoT devices with energy harvesting. IEEE Trans. Veh. Technol. **68**(2), 1930–1941 (2019). https://doi.org/10.1109/TVT.2018.2890685
30. Mnih, V., et al.: Playing Atari with deep reinforcement learning. CoRR abs/1312.5602 (2013). http://arxiv.org/abs/1312.5602
31. Morales, M., Isbell, C.: Grokking deep reinforcement learning. Manning (2020)

32. Nijimbere, D., Zhao, S., Gu, X., Esangbedo, M.O., Dominique, N.: Tabu search guided by reinforcement learning for the max-mean dispersion problem. J. Industr. Manage. Optim. **17**(6), 3223–3246 (2021)
33. Pathan, S., Shrivastava, V.: Reinforcement learning for assignment problem with time constraints (2021)
34. Perkin, T.M., Mini, S.: Assignment of IoT nodes to edge computing devices in internet of things. In: 2019 European Conference on Networks and Communications (EuCNC), pp. 528–532 (2019). https://doi.org/10.1109/EuCNC.2019.8802058
35. Perron, L., Furnon, V.: Or-tools. https://developers.google.com/optimization/
36. Rajashekar, K.: Reinforcement learning for minimizing communication delay in edge computing. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 1270–1271. IEEE (2022)
37. Rajashekar, K., Paul, S., Karmakar, S., Sidhanta, S.: Topology aware cluster configuration for minimizing communication delay in edge computing. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pp. 1310–1311. IEEE (2022)
38. Schempp, P., Preuß, K., Tröger, M.: About the correlation between crude oil corrosiveness and results from corrosion monitoring in an oil refinery. Corrosion **72**(6), 843–855 (2016)
39. Sheng, M., Dai, Y., Liu, J., Cheng, N., Shen, X., Yang, Q.: Delay-aware computation offloading in NOMA MEC under differentiated uploading delay. IEEE Trans. Wireless Commun. **19**(4), 2813–2826 (2020). https://doi.org/10.1109/TWC.2020.2968426
40. Shoham, Y., Powers, R., Grenager, T.: Multi-agent reinforcement learning: a critical survey. Technical report (2003)
41. Silver, D.: Lectures on reinforcement learning (2015). https://www.davidsilver.uk/teaching/
42. Song, Y., Yau, S.S., Yu, R., Zhang, X., Xue, G.: An approach to QoS-based task distribution in edge computing networks for IoT applications, pp. 32–39. Institute of Electrical and Electronics Engineers Inc. (2017). https://doi.org/10.1109/IEEE.EDGE.2017.50
43. Spatharakis, D., et al.: A scalable edge computing architecture enabling smart offloading for location based services. Pervasive Mob. Comput. **67**, 101217 (2020). https://doi.org/10.1016/j.pmcj.2020.101217, https://www.sciencedirect.com/science/article/pii/S1574119220300778
44. Sudharsan, B., Breslin, J.G., Ali, M.I.: Edge2Train: a framework to train machine learning models (SVMs) on resource-constrained IoT edge devices. In: Proceedings of the 10th International Conference on the Internet of Things, pp. 1–8 (2020)
45. Sun, X., Ansari, N.: Latency aware workload offloading in the cloudlet network. IEEE Commun. Lett. **21**(7), 1481–1484 (2017). https://doi.org/10.1109/LCOMM.2017.2690678
46. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge (2018)
47. Wei, Z., Jiang, H.: Optimal offloading in fog computing systems with non-orthogonal multiple access. IEEE Access **6**, 49767–49778 (2018). https://doi.org/10.1109/ACCESS.2018.2868894
48. Wikipedia: Nearest neighbor search—Wikipedia, the free encyclopedia (2022)
49. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)

50. Xia, Q., Ye, W., Tao, Z., Wu, J., Li, Q.: A survey of federated learning for edge computing: Research problems and solutions. High-Confidence Comput. **1**(1), 100008 (2021). https://doi.org/10.1016/j.hcc.2021.100008, https://www.sciencedirect.com/science/article/pii/S266729522100009X
51. Yousefpour, A., Ishigaki, G., Jue, J.P.: Fog computing: towards minimizing delay in the internet of things. In: 2017 IEEE International Conference on Edge Computing (EDGE), pp. 17–24 (2017). https://doi.org/10.1109/IEEE.EDGE.2017.12
52. Zhu, R., Liu, B., Niu, D., Li, Z., Zhao, H.V.: Network latency estimation for personal devices: a matrix completion approach. IEEE/ACM Trans. Netw. **25**(2), 724–737 (2017). https://doi.org/10.1109/TNET.2016.2612695
53. Zinonos, Z., Vassiliou, V., Ioannou, C., Koutroullos, M.: Dynamic topology control for WSNs in critical environments. In: 2011 4th IFIP International Conference on New Technologies, Mobility and Security, pp. 1–5 (2011). https://doi.org/10.1109/NTMS.2011.5720652