Information Sciences 537 (2020) 116–131

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment

Zhao Tong [a,*], Xiaomei Deng [a], Feng Ye [a], Sunitha Basodi [b], Xueli Xiao [b], Yi Pan [b]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha 410012, China
[b] Department of Computer Science, Georgia State University, Atlanta 30302, USA

ARTICLE INFO

Article history:
Received 27 November 2019
Received in revised form 10 April 2020
Accepted 13 May 2020
Available online 4 June 2020

Keywords:
Deep reinforcement learning
Energy consumption
Mobile edge computing
Response time
Task offloading

ABSTRACT

With the popularity of smart mobile equipment, the amount of data requested by users is growing rapidly. The traditional centralized processing method represented by the cloud computing model can no longer satisfy the effective processing of large amounts of data. Therefore, the mobile edge computing (MEC) is used as a new computing model to process the big growing data, which can better meet the service requirements. Similar to the task scheduling problem in cloud computing, an important issue in the MEC environment is task offloading and resource allocation. In this paper, we propose an adaptive task offloading and resource allocation algorithm in the MEC environment. The proposed algorithm uses the deep reinforcement learning (DRL) method to determine whether the task needs to be offloaded and allocates computing resources for the task. We simulate the generation of tasks in the form of Poisson distribution, and all tasks are submitted to be processed in the form of task flow. Besides, we consider the mobility of mobile user equipment (UE) between base stations (BSs), which is closer to the actual application environment. The DRL method is used to select the suitable computing node for each task according to the optimization objective, and the optimal strategy for solving the objective problem is learned in the algorithm training process. Compared with other comparison algorithms in different MEC environments, our proposed algorithm has the best performance in reducing the task average response time and the total system energy consumption, improving the system utility, which meets the profits of users and service providers.

1. Introduction

With the rapid development of the Internet of Things and wireless networks, the amount of data generated by mobile user equipments (UEs) on the edge of the network is growing rapidly. The server-side architecture with the cloud computing model as the core has been unable to meet the requirements of big data processing, mainly including latency, energy consumption, and data security requirements [1,2]. In order to solve the problem of excessive network transmission pressure and low real-time data processing, the mobile edge computing (MEC) model is proposed as a new distributed computing model [3]. The traditional cloud computing is a centralized distributed computing model, tasks generated by mobile UEs are submitted to the cloud center with sufficient computing resources for processing, latency and transmission costs are incurred due to network congestion during task submission [4–6]. Compared to the cloud computing model, the MEC model

* Corresponding author.
E-mail address: tongzhao@hunnu.edu.cn (Z. Tong).

https://doi.org/10.1016/j.ins.2020.05.057
0020-0255/

provides computing services near the edge of mobile UEs, which can respond to user requests faster and save more network transmission overhead. The MEC essentially provides storage, calculation, and other services at the edge layer close to the data generation source for users [7].

Recently, with the rapid development of computer and communication technology, the MEC model has been applied in many scenarios, such as smart city, smart home, cloud game, healthcare, etc. [8,9]. In general, these applications that require a high quality of experience (QoE) are expected to achieve low latency and energy consumption. The computing resources in the MEC model include mobile UEs and MEC servers. Whether data generated by mobile UEs is offloaded and computing resource allocation strategy plays a crucial role in improving QoE [10,11]. In the MEC model, data offloading refers to the transfer of data from the mobile UE to the MEC server, which solves the problem of insufficient local computing power and saves the power consumption of the mobile UE. Task offloading and resource allocation is a non-deterministic polynomial (NP)-hard problem [12]. In task processing, the first is to determine whether the task needs to be offloaded. If the offload strategy decides not to offload the task, the task is processed locally. If the strategy decides to offload the task, then determine which MEC server used to process the task (i.e., the process of resource allocation).

In this paper, since the process of task offloading and resource allocation in the MEC environment is essentially a process of selecting computing node for the task, we model this process as a Markov decision process (MDP) [13]. In general, the reinforcement learning (RL) method [14] is suitable for solving the MDP problem. The goal of the RL method is to learn to obtain an optimal strategy for solving the objective problem, and adaptively adjust the strategy according to the reward value of the environmental feedback during the learning process [13,15]. However, due to the poor learning effect of traditional RL methods in the complex environment, we use the deep reinforcement learning (DRL) method [16] that combines RL and deep learning (DL) [17] to solve the problem of task offloading and resource allocation. We assume that tasks generated by mobile UEs form a task flow, and we use the DRL method to select a suitable computing node for the task in the current task environment, i.e., the task offloading and resource allocation. The proposed algorithm has a good performance in optimizing the average response time and total energy consumption of the MEC system, which improves the user's QoE and reducing the system energy consumption. The main contributions of this paper are as follows:

- A DRL-based task offloading and resource allocation algorithm in the MEC environment is proposed. The algorithm adaptively learns the strategy of selecting suitable computing nodes.
- According to the consideration of the profits of users and service providers, the proposed algorithm optimizes the task average response time and the total system energy consumption.
- In order to obtain a better learning effect, the entropy weight method in the objective weighting method is used to calculate the reward value of the environmental feedback to the agent.
- In order to make the experimental environment more realistic, the mobility of UEs is considered. Experimental results show that our proposed algorithm has good performance in reducing average response time and total energy consumption.

The remainder of the paper is organized as follows. Section 2 presents the related works on solving the problem of task offloading and resource allocation in the MEC system. Section 3 describes the MEC system model, including task and task generation, local computing, and MEC server computing models. Section 4 gives the algorithm evaluation index and formulates the objective optimization problem. Section 5 introduces the theoretical background related to the proposed algorithm and describes the details of the proposed algorithm. Section 6 gives experimental parameters, compares, and analyzes the experimental results of different algorithms. Section 7 concludes this paper and proposes directions for future work.

2. Related work

In this section, we give and analyze some related works on task offloading and resource allocation in the MEC system. The purpose of task offloading is to reduce the pressure of local computing, reduce the response time of tasks, and extend the battery life of UEs [18–21]. By using the task offloading and resource allocation strategy, the task is assigned to the suitable MEC server to be executed, which can reduce the transmission and calculation energy consumption, improve the overall system utility.

Mao et al. [22] proposed a task offloading algorithm called the Lyapunov optimization-based dynamic computation offloading (LODCO), which obtains the optimal solution of the objective problem in each time slot, i.e., the task offloading strategy. The LODCO algorithm optimizes the problem of reducing task execution cost and verifies the effectiveness of the algorithm through simulation results. Liu et al. [23] proposed a job scheduling framework based on the cooperative game (COOPER-SCHED), which is used to optimize the problem of maximizing the number of jobs that meet the deadline in the MEC model, improve the service quality of the system. Samanta et al. [24] considered the latency oblivious of different services and proposed a distributed task scheduling algorithm in the MEC model, which reduces the service latency, increases the throughput and improves the overall performance of the system. Chen et al. [25] proposed a heuristic algorithm to solve the task assignment problem in the Ad-hoc mobile cloud computing (Ad-hoc MCC) model. The simulation results show that the algorithm has a good performance in reducing task response time. Wang et al. [26] proposed a new framework (MEC-WPT) combined with wireless power transfer technology in the MEC model, optimized the total energy consumption min-

imization problem under latency constraints through resource allocation algorithm. Hu et al. [27] proposed a game-based computation offloading (GCO) algorithm to solve the task offloading problem, and experimental results show that the algorithm is effective in maximizing the number of tasks completed before the deadline and reducing the energy consumption. Cao et al. [28] proposed a new task offloading algorithm to reduce energy consumption while meeting the deadline of user tasks. These research works have a good performance in solving the problem of task offloading and resource allocation in the MEC model, but they only considered the optimization of a single objective, such as latency, response time, energy consumption, etc. Since UEs and MEC servers need to be considered together in the MEC model, optimization for a single objective cannot effectively improve the system utility. Therefore, the overall performance improvement of the system should be considered more when designing the strategy.

Chen and Hao [29] formulated the problem of task offloading in the MEC model as a non-linear program, and proposed an offloading algorithm to reduce the task delay and improve the battery life of UE. However, the research work does not consider the mobility of users in the MEC simulation environment, which lacks the dynamics and complexity in the real MEC environment. Cao et al. [30] proposed a novel task offloading algorithm (TaSRD), which used the time model and energy model to formulate optimization metrics. The experimental results show that the algorithm has a good effect in reducing task completion time and energy consumption, but they did not consider the energy optimization on the MEC server. Zhang et al. [31] proposed a task offloading algorithm based on the consideration of energy consumption and latency, which is an iterative search algorithm. The proposed algorithm obtains a lower weighted sum of energy consumption and latency than other comparison algorithms in numerical experiments, verifies the effectiveness of the proposed algorithm. However, the research work only made numerical solutions in the experiment, which lacks certain reality; the evaluation index used the weighted sum of energy consumption and latency, instead of reducing energy consumption and delay at the same time. Dinh et al. [32] proposed a task offloading optimization framework in the MEC model, which considers reducing task latency and the energy consumption of UE. And the SDR-based algorithm is proposed for objective optimization, and the simulation results show that the algorithm has better performance than other comparison algorithms. However, the algorithm just only considers the UE when optimizing the energy consumption, but no consider the MEC server; the total cost of latency and energy consumption is used as the evaluation index of algorithm performance, which does not accurately reflect the optimization results of the two objectives. Mao et al. [33] considered optimizing two critical objectives in the MEC system, including power consumption of UEs and task execution delay, and proposed an online task offloading algorithm. The experimental results show that the algorithm is effective for multi-objective optimization and improves the user's QoE. The research work used a small number of mobile devices in the experiment, which lacked practicality; and it only considered reducing the energy consumption of UEs, without considering the overall system cost.

At present, most researches on task offloading and resource allocation in the MEC model is based on simple traditional methods. Although such methods effectively solve the problem of task offloading and resource allocation, they cannot adapt to complex MEC environments, and the algorithm is not stable, the experimental effect is improved limit. With the development of computer software and hardware and the improvement of related technologies, machine learning methods are used to solve optimization problems such as task offloading and resource allocation, bringing new ideas to relevant research work, and the effect of objective optimization is greatly improved. Wang et al. [34] proposed a task scheduling algorithm based on reinforcement learning, which is used to balance the reduction of energy consumption and delay. Experimental results show that the proposed algorithm performs better than other comparison algorithms. Ning et al. [35] proposed an intelligent task offloading system in the edge computing environment, which used the matching method and DRL method to solve the task offloading and resource allocation problem, improve the user's QoE and system utility. Huang et al. [36] considered the problem of task offloading in multi-users MEC system, and proposed a task offloading and resource allocation scheme based on DRL method. Experimental results show that the proposed scheme has a good performance in reducing the total system cost of delay and energy consumption.

To the best of our knowledge, the existing machine learning-based task offloading and resource allocation schemes in the MEC environment has some shortcomings, such as not consider the user's mobility in the system framework, the number of UEs and tasks is small, the algorithm design is complex, and the optimization objective is single, etc. In this paper, we propose a novel DRL-based online algorithm (DTORA) for solving task offloading and resource allocation problems in the MEC environment. The proposed algorithm makes decisions for task offloading and resource allocation at the same time, the algorithm design is more optimization. When designing the environment framework, we considered the user's mobility, which is closer to the actual application scenarios. In addition, the profit of users and server providers are considered in the optimization of the objective.

3. Mobile edge computing system model

In this section, we consider a MEC system with multi-mobile UE and multi-server, as shown in Fig. 1. In the MEC system model, it mainly includes mobile UEs, base stations (BSs), servers, and wireless networks. The task generated by the mobile UE can be executed locally or offloaded to the MEC server through the wireless network. The mobile UE has mobility, and the time interval of the user movement is subject to an exponential distribution. The BS and servers form the MEC server, the BS is used to receive task data, and the server is used to process tasks. In order to close to the real environment, we consider that

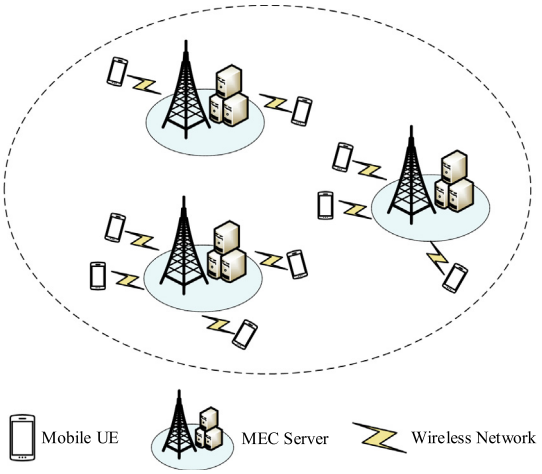

Fig. 1. The MEC system with multi-mobile UE and multi-server.

MEC servers are heterogeneous. MEC servers are distributed in several locations in the MEC system for receiving and processing nearby user tasks, reducing task response time and extending the battery life of the mobile UE.

3.1. Task and task generation models

In the MEC system, tasks are generated by UEs, and the time interval of task generation is subject to a Poisson distribution [37]. The UE k generates n_k independent tasks, defined as $N_k = \{1, 2, \ldots, n_k\}$. The task has several important attributes, defined as $task = \{id_u, id_t, sub_t, d_t, c_t, l_t, mem_t, cpu_t\}$, where id_u presents the id of the mobile UE that generated the task, id_t presents the task id, sub_t presents the time that the mobile UE submits the task, d_t (in bits) presents the task data size, c_t (in CPU cycles/bit) presents CPU cycles to compute one bit data of task, l_t (in CPU cycles) is the total CPU cycles of task and $l_t = d_t c_t$, mem_t and cpu_t presents the memory and CPU resources required for task calculation, respectively. We assume that each task has a submission time, and the tasks are queued according to the submission time, the earlier the task is submitted, the sooner it is processed. Due to the mobility, the UE may be near different BSs at different times, so tasks generated by the mobile UE at different times may be offloaded to MEC servers on different BSs for execution.

3.2. Local computing model

Since the mobile UE has computing ability, tasks can be processed locally if the mobile UE has sufficient resources (memory and CPU), and we assume that the mobile UE can only process one task at a time. The CPU frequency of the local equipment k is f_k, which is the CPU cycles per second of the mobile UE k, reflecting its computing ability. The processing time of the local computing model only includes the computation time and no transmission time, the time cost of task i generated by mobile UE k using local processing is defined as follows:

$$t_{i,l} = \frac{l_i}{f_k} \tag{1}$$

In addition, the power consumption and energy consumption of task i on mobile UE k are expressed as Eq. (2) and Eq. (3), respectively.

$$P_{i,k} = \kappa (f_k)^3 \tag{2}$$

$$E_{i,k} = \kappa (f_k)^2 l_i \tag{3}$$

where κ is the effective switching capacitance in the chip [33].

3.3. MEC server computing model

The number of tasks generated by mobile UEs is large. Because the resources of local computing are limited, not all tasks can be executed locally. Therefore, tasks can be offloaded to MEC servers for processing when local resources are insufficient. The processing time of the MEC server computing model includes the transmission time and computation time, and the transmission time refers to the time cost of uploading task data from the mobile UE to the MEC server. In order to calculate the communication time, the communication rate [38] of the UE k to BS m is defined as:

$$r_{k,m} = W\log_2\left(1 + \frac{p_k g_{k,m}}{N_0 W}\right) \tag{4}$$

where W is the communication bandwidth between UE k and BS m, p_k is the transmit power of the UE k, N_0 is the noise power spectral density of the BS $m; g_{k,m}$ is the channel gain between UE k and BS m, and $g_{k,m} = (D_{k,m})^{-\sigma}, D_{k,m}$ is the distance between UE k and BS m, σ is the path-loss exponent.

The time cost of the task offloading to the MEC server for processing is the sum of upload time and calculation time. Therefore, the time cost of task i generated by UE k offloading to the server j on the BS m for processing is defined as:

$$t_{i,off} = \frac{d_i}{r_{k,m}} + \frac{l_i}{f_{m_j}} \tag{5}$$

where f_{m_j} is the CPU frequency of the serve j on the BS m.

The energy consumption of the task offloading to the MEC server for processing is the sum of upload energy consumption and calculation energy consumption. Therefore, the energy consumption of task i generated by UE k offloading to the server j on the BS m for processing is defined as:

$$E_{i,off} = p_k \frac{d_i}{r_{k,m}} + d_i q_{m_j} \tag{6}$$

where q_{m_j} is the energy consumption of each bit of data calculated by the server j on the BS m.

4. Problem formulation

In this section, we first formulate indicators for evaluating the algorithm performance, including average task response time, total energy consumption, and system utility, then formulate the optimization problem of this paper.

4.1. Optimization indicators

The task response time refers to the time interval from the task submission time to the task completion time, which is determined by the task queuing time, communication time, and calculation time. Since the data size of task results returned is small, the transmission time only considers the time cost of offloading the task data from the user device to the MEC server, and the time cost of the result return is negligible. If the task is calculated locally, the communication time is 0. We first consider the response time of each task and then calculate the average response time of all tasks, the average response time reflects the user's QoE. We defined the response time and the average response time of tasks in Eq. (7) and Eq. (8) respectively.

$$res_i = fin_i - sub_i \tag{7}$$

$$res_{avg} = \sum_{i=1}^{n} \frac{res_i}{n} \tag{8}$$

where fin_i is the finish time of the task i.

Energy consumption includes local computing and offloading computing energy consumption. The total energy consumption is the sum of all task energy consumption, so defined as follows:

$$E_{total} = \sum_{i=1}^{n} E_i \tag{9}$$

where $E_i = \begin{cases} E_{i,k}, & \text{local computing} \\ E_{i,off}, & \text{offloading computing} \end{cases}$.

In this paper, considering the benefits of mobile users and service providers, we optimize the problem of minimizing time and energy costs. We use the average response time and total energy consumption as the evaluation indicators of the algorithm. In order to reflect the performance improvement of the optimization objective relative to local processing, we use the system utility as the evaluation metric. The system utility is the weighted sum of task response time and energy consumption relative to local processing improvement, and defined as follows:

$$U = \sum_{i=1}^{n} \lambda_{res} \frac{res_i^l - res_i}{res_i^l} + \lambda_E \frac{E_i^l - E_i}{E_i^l} \tag{10}$$

where $\lambda_{res}, \lambda_E \in [0, 1]$, and $\lambda_{res} + \lambda_E = 1; res_i^l$ and E_i^l are the response time and energy consumption of the task executed locally, respectively.

4.2. Optimization problem

In this paper, we consider minimizing the average task response time and total energy consumption in the MEC environment, improving the user's QoE, and saving system power consumption. The execution of the task on the computing node (including UEs and MEC servers) is limited by some resource conditions, and the task can be executed only when the conditions are met. We assume that the computing node has the limit of the number of parallel tasks executed at the same time, and the maximum number of parallel tasks is defined as δ. If the number of tasks executed at the same time on the compute node is less than δ, the new task can be received; otherwise, the new task needs to wait for the release of resources. In addition, task execution is also limited by memory and CPU resources. Only when the total memory and CPU resources occupied by all tasks are less than the total resources of the computing node, new tasks can be executed; otherwise, the new task will have to wait for resources to be released. The objective optimization problem can be formulated as:

$$\begin{aligned} \min_{\{y\}} \quad & res_{avg} \text{ and } E_{total} \\ \text{s.t.} \quad & y \leqslant \delta \\ & \textstyle\sum_{i=1}^{y} mem_i \leqslant M \\ & \textstyle\sum_{i=1}^{y} cpu_i \leqslant C \end{aligned} \tag{11}$$

where y is the number of tasks executed in parallel on the computing node, M and C are the memory and CPU capabilities of the computing node, respectively.

The commonly used terms in the DTORA model are shown in Table 1.

5. Adaptive task offloading and resource allocation algorithm

In this section, we first introduce the relevant theoretical background knowledge of the DRL method; second, we give some important settings in our proposed algorithm; finally, we introduce the design details of our algorithm.

5.1. Theoretical background

Deep Q-Network (DQN) [39] is a specific algorithm in the DRL method, which is combined by Q-learning algorithm in RL method and convolution neural network (CNN) [40]. Q-learning is an adaptive algorithm without guidance and supervision, which can learn knowledge from the interaction with the environment. An MDP can be described using a quadruple s, a, r, s', where s is the current environment, a is an action, r is the reward value and s' is the new environment. The Q-learning algorithm is used to solve the MDP problem, the agent selects an action in the current environment to enter a new environment, and the environment feedbacks a corresponding reward value according to the quality of the selected action. Q-learning algorithm updates the action value function (Q-value) according to the selection of each action. After a certain number of iterations can reach a convergence state, and the agent can select a better action. The update formula of the Q-learning algorithm is defined as follows:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\left(r + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a)\right) \tag{12}$$

where $Q_t(s,a)$ and $Q_{t+1}(s,a)$ are the Q-value in the current environment and the new environment, respectively. α $(0 \leqslant \alpha \leq 1)$ is the learning rate, which represents the degree of the agent pays attention to the new and old experience information. The value of α is 0 indicates that the agent is only attention to the old experience information, and the value of α is 1 indicates that the agent is only attention to the new experience information. γ $(0 \leqslant \gamma \leq 1)$ is the discount factor, which represents the degree of the agent pays attention to the current and future reward. The value of γ is 0 indicates that the agent is only attention to the current reward, and the value of γ is 1 indicates that the agent is only attention to the future reward.

Q-learning is a simple and effective self-learning algorithm, which is suitable for solving Markov decision problems. However, since the Q-value in the Q-learning algorithm are stored in a two-dimensional matrix (i.e., Q-table), the size of the Q-table is limited by computer memory, which may lead to the Q-table explosion problem [41]. Therefore, the Q-learning algorithm cannot solve the problem of a large number of states and actions. In order to solve the Q-table explosion problem, the neural network is used to replace the Q-table, the function fitting in the neural network is equivalent to the update of the Q-table, that is the DQN algorithm.

Compared with Q-learning, the DQN algorithm also adds some technical methods, mainly including value function approximation, experience replay, and target Q-network, these methods improve the effectiveness, stability and convergence speed of the algorithm. The value function approximation method is to approximate the Q-value by non-linear function using CNN in the DQN algorithm [42,43]. The current environment state s as the input to the CNN, training through the convolutional layer and the fully connected layer, finally outputting the Q-value of actions. The agent selects an action according to the Q-value to make the current optimal decision. The experience replay mechanism is to solve the oscillations and divergence problems caused by the correlation between experience data. The specific approach is to use an experience pool to store the transition samples (s, a, r, s'). When there are enough samples in the experience pool, randomly select sam-

ples (minibatch) to train the DQN algorithm. In order to reduce the correlation between the calculation of the target Q-value and the current Q-value, improve the stability of the algorithm, the DQN algorithm uses two CNN with the same structure but different parameters, namely the target network and the evaluation network. The target network is used to obtain the target Q-value $\widehat{Q}(s,a)$, and the evaluation network is used to evaluate the current state-action pair function value $Q(s,a)$. During the parameters update process of the neural network, the evaluation network uses the latest parameters, and the target network uses older parameters. After every ζ steps, the parameters of the evaluation network are cloned to the target network. The pseudocode of the DQN algorithm is described as follows:

Algorithm 1.: Deep Q-learning with Experience Replay

Input: State-value
Output: Action-value
Initialize replay memory Θ to capacity Ω;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta'=\theta$;
for *episode = 1, G* **do**
  Initialize sequence $s_1 = x_1$;
  for $t = 1,\ T$ **do**
    With probability ϵ, select a random action a_t, otherwise select $a_t = arg\max_a Q(s_t, a; \theta)$;
    Execute action a_t and observe reward r_t and next observation x_{t+1};
    Set $s_{t+1} = s_t,\ a_t,\ x_{t+1}$;
    Store transition $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in Θ;
    Sample random minibatch of transitions $(s_j,\ a_j,\ r_j,\ s_{j+1})$ from Θ;
    Set $y_i = \begin{cases} r_j, \text{for terminal at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta'), \text{otherwise} \end{cases}$;
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$;
    Every ζ steps, clone Q to $\hat{Q}$;
  end
end

5.2. Important settings of the DTORA algorithm

In this paper, since the optimization objective is to minimize the task response time and energy consumption at the same time, we consider using the linear weighting method to balance the optimization bias of the two objectives. Most of the research work uses a simple subjective weighting method to determine weight allocation. However, due to the influence of subjective awareness is too much, this method tends to result in inaccurate weight allocation and poor objective optimization effect. Therefore, we use the entropy weight method [44] of the objective weighting method to determine the weight allocation of response time and energy consumption. We define the number of samples in the entropy weight method as p, which is the number of actions selected in this paper; the number of indicators is q, which is the number of optimization objectives, and $q = 2$ in this paper. The calculation steps of the entropy weight method are as follows:

1. Normalization of indicators. Since the measurement units of the indicators are not uniform, the value of the indicators differ greatly. Therefore, we have to normalize the size of all indicators to an order of magnitude. We use the min–max normalization method, defined as follows:

$$x\prime_{i,j} = \frac{x_{i,j} - \min\{x_{1,j}, \ldots, x_{p,j}\}}{\max\{x_{1,j}, \ldots, x_{p,j}\} - \min\{x_{1,j}, \ldots, x_{p,j}\}} \tag{13}$$

where $i = 1, \ldots, p,\ j = 1, 2$.

2. Calculate the proportion of each sample value under various indicators, defined as follows:

$$p_{i,j} = \frac{x\prime_{i,j}}{\sum_{i=1}^{p} x\prime_{i,j}} \tag{14}$$

3. Calculate the entropy of each indicator, defined as follows:

$$e_j = -k\sum_{i=1}^{p} p_{i,j} \ln(p_{i,j}),\ \ k = \frac{1}{\ln(p)} \tag{15}$$

4. Calculate the weight of each indicator, defined as follows:

$$w_j = \frac{1 - e_j}{2 - \sum_{j=1}^{2} e_j} \tag{16}$$

When using the DQN algorithm to solve the optimization problem in this paper, we need to set some important algorithm elements according to the specific problem. The settings of these elements affect the performance of the algorithm, including the state space, action space, and reward function.

State Space. According to the optimization objective in this paper, we take the weighted sum of the response time and energy consumption of the task on each computing node as a state, and the state space is defined in Eq. (17). The weighting method uses the entropy weight method, the task response time and energy consumption are normalized (shown in Eq. (13)).

$$s = (s_1, \ldots, s_p),\ \ s_i = w_{i,1} res'_i + w_{i,2} E'_i \tag{17}$$

Action Space. In this paper, we consider solving the problem of task offloading and resource allocation in the MEC environment. Therefore, the selection of the computing node is taken as the action selection, where computing nodes refer to the mobile UE that generates the task and servers on the BS that the mobile UE is located. The action space can be represented as $a = (UE, server_1, \ldots, server_j)$, where j is the number of servers on the BS. The action value corresponding to the selected computing node is 1, and the action value corresponding to the other computing nodes is 0. Assuming the current task is executed locally, then $a = (1, 0, \ldots, 0)$.

Reward Function. Select an action in the current environment state, that is, select a suitable computing node (allocate computing resources) for the current task. If the task is assigned to the MEC server for processing, the task is offloaded; otherwise, the task is processed locally. The system environment feedbacks a reward according to the quality of the action selected by the agent. Generally, the larger the value of the reward indicates the better the action selection. In order to guide the agent to learn the experience of minimizing response time and energy consumption, we use the negative of the weighted sum of the response time and energy consumption after the task is actually executed as the reward value, defined as Eq. 18.

$$r = -(w_{i,1} res_i\prime + w_{i,2} E_i\prime),\ \ -1 \leqslant r \leqslant 0 \tag{18}$$

5.3. The design of the DTORA algorithm

In this paper, we propose a DQN-based algorithm for solving task offloading and resource allocation problem in the MEC environment, and the algorithm is mainly used to decide whether the task needs offloading and offload to which server. In order to minimize the optimization objective of average response time and total energy consumption, we consider the response time and energy consumption of the task when setting the state and reward value attributes in the algorithm. In general, the task executed on a computing node with a faster CPU frequency will generate more energy consumption, while the task calculation time is longer on a compute node with a slower CPU frequency. Therefore, the balance between task response time and energy consumption is essential to solving this optimization problem. We balance the two objectives by weighted summation and use the entropy weight method to calculate the weights of the two objectives for each task. The larger the weight value, the more attention is paid to the objective, and the degree of optimization is greater.

Mobile UEs generate tasks, and tasks wait in the task queue for execution according to the task submission time. If the algorithm decides not to offload the task, it is processed locally; otherwise, the task is offloaded to the selected server for processing. The environment feedbacks a reward based on the time cost and energy cost of task execution to guide the agent's learning and improve the accuracy of the next action selection. After a certain number of generations of training, the performance of the algorithm is improved, and the effect of objective optimization is remarkable. The pseudocode of the DTORA algorithm is described as follows:

Algorithm 2.: The DTORA Algorithm

Input: Mobile user equipments
Output: Task average response time and total energy consumption
Initialize the MEC environment;
Initialize the DRL model;
Initialize parameter settings;
Simulate UEs generate tasks in Poisson distributions form;
for *each task generated by UE* k **do**
  Determine the BS location Λ of the UE k when the task is generated;
  Obtain the current experimental environment Φ;
  Use DQN method to select a computing node for the task in the environment Φ;
  if *the selected computing node is UE* k **then**
    The task is processed locally;
  end
  else
    Offload the task to the MEC server at BS location Λ for processing;
    Calculate the time cost of the task offloading to the MEC server;
  end
  Calculate the reward value feedback to the agent according to Equation 18;
  Calculate the response time and energy consumption of the task;
  Update the optimal strategy of the DTORA algorithm;
end
return

6. Experiments and analysis

In this section, we first introduce the experimental platform and set some important experimental parameters; secondly, we introduce several algorithms for performance comparison in this paper; finally, we show and analyze the experimental results under different experimental environment settings.

6.1. Experimental settings and comparison algorithms

In this paper, the experimental platform used is python 3.6 on the Windows 10 system, using Google's TensorFlow-GPU 1.14 as the deep learning framework. The experimental parameters settings and the hyperparameter settings for DQN in our proposed algorithm are described in Table 2, and we determined the hyperparameter settings after a lot of experiments.

Table 1
Commonly used terms in DTORA model.

Notation	Definition
c_t	CPU cycles for computing one bit data
cpu_t	CPU resources for task calculation
d_t	Task data size
e	Number of tasks executed in parallel
f_{m_j}	CPU frequency of server m_j
f_k	CPU frequency of the UE k
fin_i	Finish time of task i
$g_{k,m}$	Channel gain between UE k and BS m
id_t	Task id
id_u	Id of the UE
l_t	Total CPU cycles of task
mem_t	Memory resources for task calculation
n	Total number of tasks
n_k	Number of tasks generated by UE k
p_k	Transmit power of UE k
q_{m_j}	Energy consumption of each bit data calculated by server m_j
$r_{k,m}$	Communication rate between UE k and BS m
res_{avg}	Average response time of tasks
res_i	Response time of task i
sub_t	Task submission time
$t_{i,l}$	Time cost of task i by local computing
$t_{i,off}$	Time cost of task i by MEC server computing
k	kth UE
C	CPU capabilities
E_i	Energy consumption of task i
$E_{i,off}$	Energy consumption of task i on server m_j
$E_{i,k}$	Energy consumption of task i on UE k
E_{total}	Total energy consumption of tasks
M	Memory capabilities
N_0	Noise power spectral density of BS
$P_{i,k}$	Power consumption of task i on UE k
W	Communication bandwidth between UE k and BS m
κ	The effective switching capacitance in the chip
δ	Maximum number of parallel tasks

In order to verify the performance of the proposed algorithm, we use the following algorithms for comparison. Q-Learning (QL) algorithm is a classic RL algorithm with good learning effects, which is suitable for solving combinatorial optimization problems such as task offloading and resource allocation. Round_Robin (RR) algorithm is simple and easy to implement, which assigns tasks submitted by UEs to computing nodes in turn for processing and starts again after one rotation. Since the RR algorithm does not consider the processing capacity of each computing node, it is easy to cause the load imbalance between computing nodes. Therefore, Weighted Round_Robin (WRR) algorithm is used to solve this problem. The WRR algorithm assigns different weights to each computing node according to its different processing capacity, so that computing nodes can accept service requests with the corresponding number of weight in one rotation. The Random algorithm is to randomly assign the task request to the computing node for processing, and the algorithm is simple to implement. According to probability theory, the actual effect of the Random algorithm is close to the RR algorithm as the number of tasks assigned to the computing nodes increases. The Local algorithm does not offload tasks generated by UEs, and all tasks are executed locally, which results in insufficient local resources and excessive computing pressure.

6.2. Experimental results and analysis

In order to better evaluate the performance of the algorithm, we performed experiments under different experimental environment settings, including task data size, number of UEs, and the number of BSs. We compared the performance of algorithms based on the experimental results, analyzed the effects of different experimental environments on the experimental results.

6.2.1. Task data size

Since mobile UEs generate user requests in the form of streams, the size of task data is unknown. In order to find the effect of the task data size on the objective optimization effect, we consider experimenting under different task data sizes, including 10–100 KB, 100–500 KB, 500–1000 KB and 1000–1500 KB. The experimental results of the average response time and total energy consumption for the four task data sizes are shown in Fig. 2.

Table 2
Experimental parameters and hyperparameter settings.

Parameters	Value	
c_i (cycles/bit)	500	
d_i (KB)	Unif $(10, 100)$, Unif $(100, 500)$,	Unif $(500, 1000)$, Unif $(1000, 1500)$
f_k (GHZ)	Unif $(0.5, 1.0)$	
f_{m_j} (GHZ)	Unif $(5.0, 10.0)$	
p_k (mW)	100	
q_{m_j} (J/bit)	Unif $\left(10^{-8}, 2 \times 10^{-8}\right)$	
C (%)	100	
$D_{k,m}$ (m)	randint $(50, 200)$	
M (GB)	16	
N_0 (dBm/HZ)	-174	
W (MHZ)	2.0	
ϵ	$(0, 1)$	
κ	10^{-28}	
σ	4	
Number of mobile UEs	$10, 20, 50, 100$	
Number of BSs	$4, 8, 16, 32$	
Activation function	ReLU	
Loss function	Mean-square error	
Mini-Batch size	32	
Optimizer	Adam	
Replay memory	100	
γ	0.7	
α	10^{-4}	

The Fig. 2(a)–(d) represent the experimental results of the task data size from small to large. The left vertical axis and the right vertical axis in the subgraph are the experimental results of average response time and total energy consumption, respectively, and the horizontal axis represents comparison algorithms. First, we can see that the average response time and total energy consumption increase as the task data size increases. This is because the larger the task data size, the longer the processing time of the task on the computing node and the transmission time of the task on the transmission channel; similarly, the more the energy consumption for processing the task with the larger data size. Second, we see that the average response time and total energy consumption of the DTORA algorithm are minimal under the four task data sizes. In general, the experimental results of the DTORA algorithm and the QL algorithm are optimal, because the two algorithms can adaptively learn to make decisions, and have good effects on solving complex combinatorial optimization problems. However, the performance of the DTORA algorithm is better than the QL algorithm, because the DQN algorithm used by the DTORA algorithm is a combination of the QL method and the RL method, the DTORA algorithm has a better learning effect than the traditional QL algorithm. Among the WRR, RR, and Random algorithms, the experimental results of the WRR algorithm are optimal. This is because compared to the RR algorithm, the WRR algorithm considers the difference in computing capability between computing nodes. By assigning weights to computing nodes, the computing nodes with more computing capability can process the more tasks, balance the load between computing nodes. Therefore, the task response time is significantly reduced, and the energy consumption of the task is reduced to a certain extent. The RR and Random algorithms have similar experimental effects. With the increase of random selection times, the effect of the Random algorithm is close to the RR algorithm. Finally, we see that if tasks are not offloaded to MEC servers for processing, but all tasks are processed by mobile UEs, the experimental results are the worst. This is due to the locally computing has limited computing resources and low computing capability, resulting in the task waiting for processing time and processing time to increase. And as the task data size increases, the average response time and energy consumption of the local algorithm increase faster. It can be known that the task data size has a great influence on the average response time and total energy consumption. Therefore, selecting the suitable computing node according to the task data size has an important role in the optimization objective.

In this paper, the system utility reflects the overall optimization effect of the algorithm on the MEC system. The larger the system utility value, the better the performance of the algorithm. The system utility results of the six algorithms under different task data sizes are shown in Fig. 3.

According to the comparison between algorithms, DTORA algorithm has the best performance for improving system utility, and can better consider the profits of users and service providers. The performance of the QL algorithm is second only to the DTORA algorithm. The optimization performance of the WRR algorithm is third, and the result is much worse than the QL algorithm, the experimental results of the RR and Random algorithm are similar. Due to the calculation of system utility is to consider the improvement of the actual task response time and energy consumption relative to the local processing, the system utility of the local algorithm is 0. In this experiment, we can know that the task data size has no significant impact on the system utility value, and the overall optimization effect of the system depends on the performance of the algorithm.

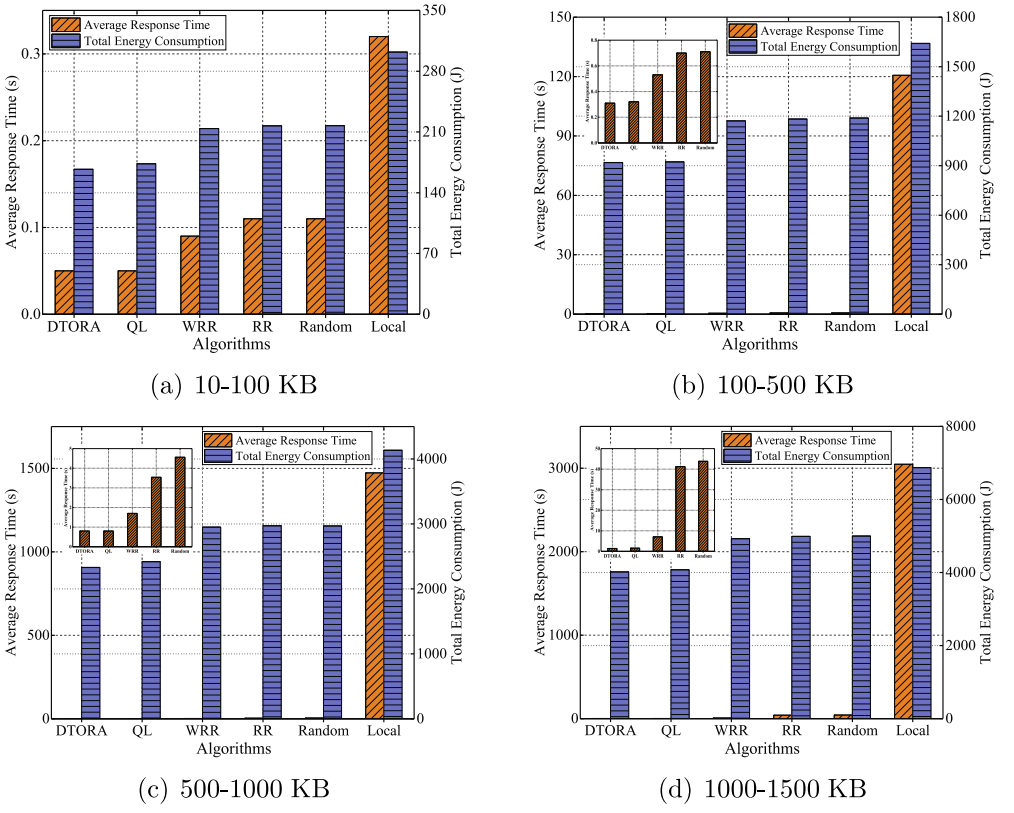

Fig. 2. Comparison of average response time and total energy consumption for different task data sizes.

6.2.2. Number of UEs

In this experiment, we show the experimental results of the average response time and total energy consumption of the six algorithms under different number of UEs, including 10 UEs, 20 UEs, 50 UEs, and 100 UEs. The experimental results are shown in Fig. 4.

Fig. 4(a)–(d) represent the experimental results of the number of UEs from small to large. We see that as the number of UEs increases, the average response time of each algorithm is in a range, and the total energy consumption is increasing. This is because the increase in the number of UEs leads to an increase in the number of tasks generated by UEs, which directly affects the total energy consumption of tasks. The task average response time is not significantly affected, which is because

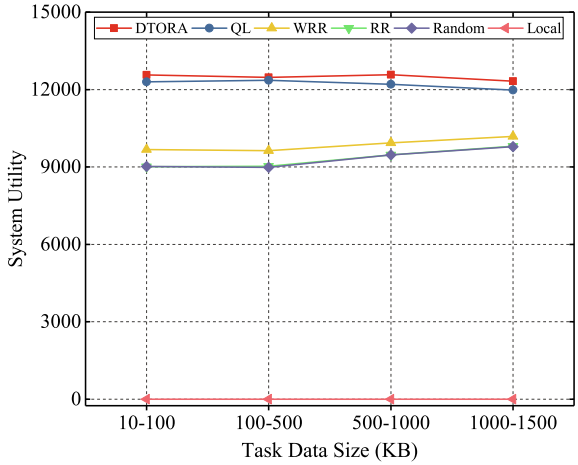

Fig. 3. Comparison of system utility for different task data sizes.

the number of tasks generated by each UE is roughly the same in the same time period, and the range of task data size is also the same. Therefore the task average response time is similar under different number of UEs. The experimental results of the DTORA algorithm are optimal compared with other comparison algorithms, and the optimization effect of the QL algorithm is inferior to the DTORA algorithm but better than other comparison algorithms (excluding the DTORA algorithm). The WRR algorithm has an obvious effect on the optimization of the task average response time compared to the RR algorithm. The RR and Random algorithms have roughly the same effect on objective optimization. The performance of the local algorithm is the worst in all comparison algorithms. It can be known that the change in the number of UEs has a significant impact on the total energy consumption, but has no significant impact on the task average response time.

The system utility results of the six algorithms under different number of UEs are shown in Fig. 5. First, we can see that the value of system utility increases as the number of UEs increases. This is because the system utility value is the sum of the offloading utility of each task. The increase in the number of UEs leads to an increase in the number of tasks. Therefore the system utility value increases with the number of UEs increase. Second, the experimental results of the DTORA algorithm under different number of UEs are optimal. The system utility value of the QL algorithm is close to the DTORA algorithm when the number of UEs is small; however, as the number of UEs increases, the system utility value of the QL algorithm is significantly lower than the DTORA algorithm. This is because the complexity of the MEC system increases as the number of UEs increases, and the learning effect of the DTORA algorithm is better than the QL algorithm in a complex environment. In this experiment, we can know that the number of UEs and the performance of the algorithm both have a significant impact on the system utility value.

6.2.3. Number of BSs

The number of BSs represents the computing capability of the MEC system. The more the number of BSs, the more MEC servers, and the distribution of BSs is wider. In this experiment, we compare the experimental results of the average response time and total energy consumption of the six algorithms under different number of BSs, including 4 BSs, 8 BSs, 16 BSs and 32 BSs. The experimental results are shown in Fig. 6.

Fig. 6(a)–(d) represent the experimental results of the number of BSs from small to large. It can be seen from Fig. 6 that as the number of BSs increases, the average response time and total energy consumption of each algorithm has no obvious change. This is because under the four number of BSs of this experiment, the computational resources required for tasks are sufficient. Thus the change in the number of BSs has no significant effect on the experimental results. Similar to the

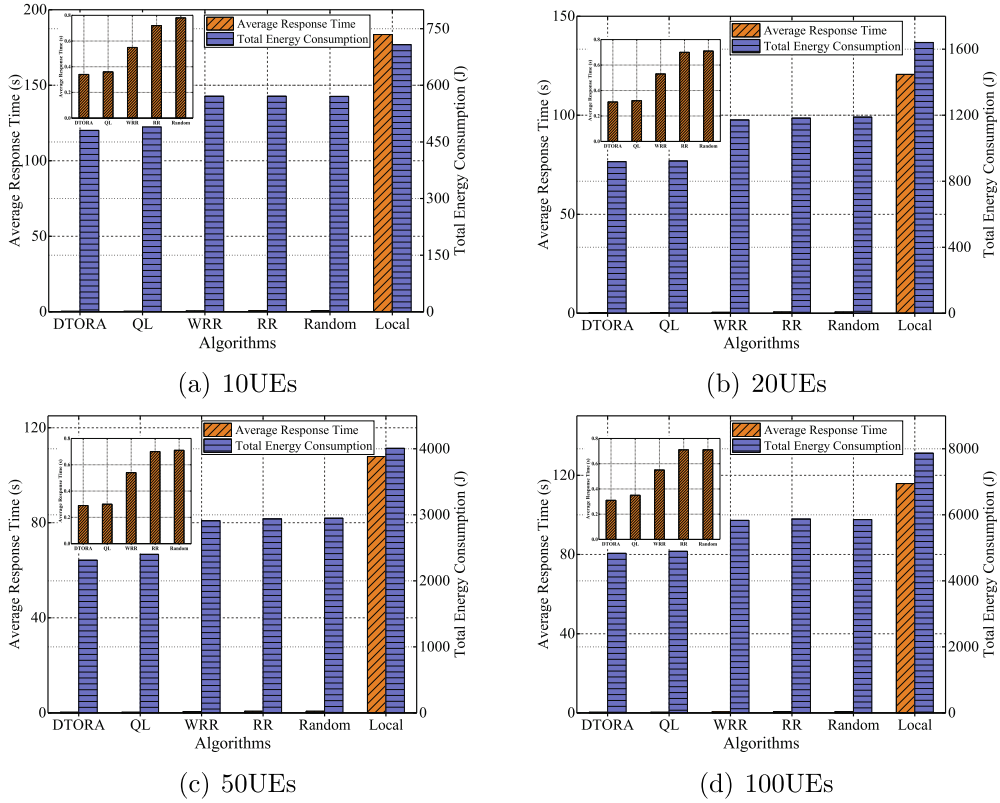

Fig. 4. Comparison of average response time and total energy consumption for different number of UEs.

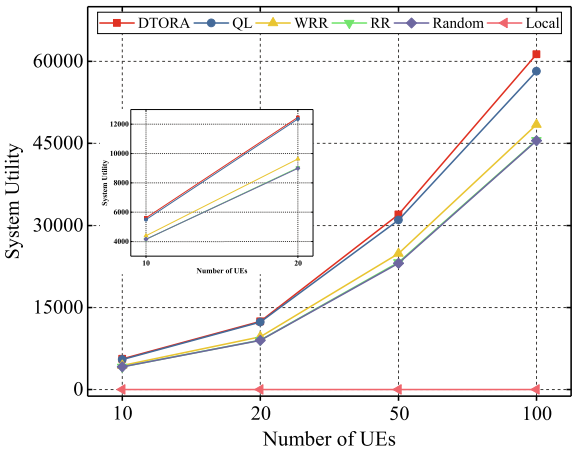

Fig. 5. Comparison of system utility for different number of UEs.

results of experiments with task data size and number of UEs, according to the optimization performance of the algorithm, the six algorithms are listed in order from good to poor: DTORA, QL, WRR, RR, Random and Local. In this experiment, we can know that the number of BSs has no significant impact on reducing average response time and total energy consumption when the computing resources of the MEC system are sufficient.

The system utility results of the six algorithms under different number of BSs are shown in Fig. 7. Similar to the experimental results in Fig. 3, the system utility values of each algorithm are almost the same under different number of BSs, which indicates that the change of the number of BSs has no significant influence on the system utility value. This is because in the experiment of the number of BSs in this paper, the change in the number of BSs has no effect on the task average response time and total energy consumption, and the number of UEs has not changed. Therefore the system utility value

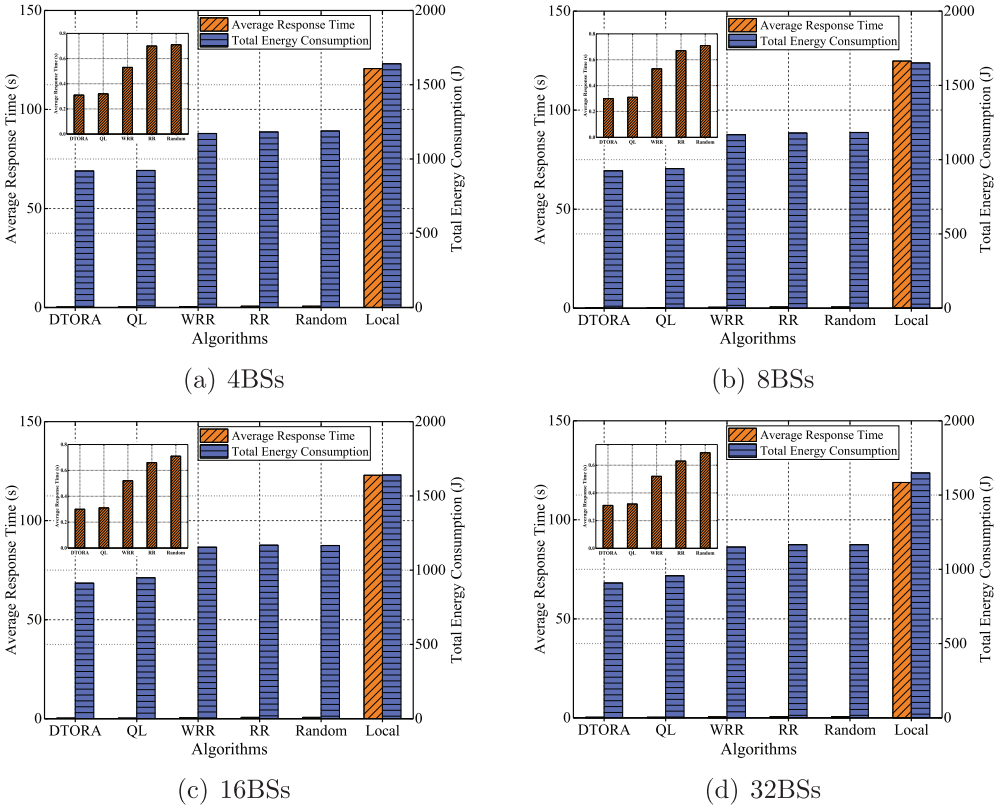

Fig. 6. Comparison of average response time and total energy consumption for different number of BSs.

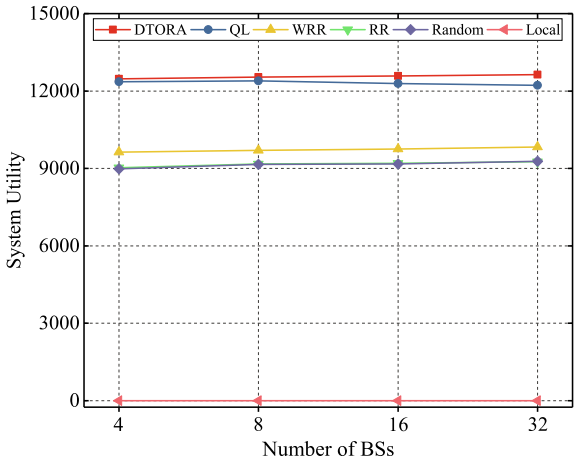

Fig. 7. Comparison of system utility for different number of BSs.

under different number of BSs has not changed much. The system utility results of the DTORA algorithm are optimal, and the suboptimal is the QL algorithm. The WRR algorithm is superior to the other three comparison algorithms, and the experimental results of RR and Random algorithms are roughly the same. It can be known that the number of BSs has no significant influence on the system utility.

7. Conclusion

In this paper, we consider solving the problem of task offloading and resource allocation in the MEC environment of multi-mobile UE and multi-server. A new adaptive algorithm is proposed to reduce the task average response time and the total system energy consumption. We model the objective problem as an MDP, and based on the intelligent learning ability of the DRL method, select the suitable computing node for each task, and complete the task offloading and resource allocation process. Moreover, in order to achieve a better learning effect of the algorithm, we use the entropy weight method as a method to calculate the weight in the objective weighted sum, avoiding the interference caused by human subjectivity. The experimental results verify the effectiveness of the DTORA algorithm, reduce the time and energy cost in the task processing, and improve the user's QoE and the overall system utility. In the future work, we intend to solve the problem of task offloading and resource allocation under the collaborative computing model that combines MEC with traditional cloud computing, and better utilize and improve the existing computing model to improve the system service level.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the Program of National Natural Science Foundation of China (Grant Nos. 61502165, 61602170), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202).

References

[1] C. Esposito, A. Castiglione, F. Pop, K.-K.R. Choo, Challenges of connecting edge and cloud computing: a security and forensic perspective, IEEE Cloud Comput. 4 (2) (2017) 13–17.
[2] K. Li, X. Tang, K. Li, Energy-efficient stochastic task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2013) 2867–2876.
[3] T.Q. Dinh, J. Tang, Q.D. La, T.Q. Quek, Offloading in mobile edge computing: Task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (8) (2017) 3571–3584.
[4] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2016) 919–933.
[5] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization, Soft Comput. 23 (21) (2019) 11035–11054.
[6] K. Li, X. Tang, B. Veeravalli, K. Li, Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems, IEEE Trans. Comput. 64 (1) (2013) 191–204.
[7] S. Wang, Y. Zhao, J. Xu, J. Yuan, C.-H. Hsu, Edge server placement in mobile edge computing, J. Parallel Distrib. Comput. 127 (2019) 160–168.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[9] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: a survey, IEEE Internet Things J. 5 (1) (2017) 450–465.
[10] P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading, IEEE Commun. Surveys Tutorials 19 (3) (2017) 1628–1656.
[11] L. Jiao, H. Yin, H. Huang, D. Guo, Y. Lyu, Computation offloading for multi-user mobile edge computing, in: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, 2018, pp. 422–429..
[12] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Network. 24 (5) (2015) 2795–2808.
[13] Z. Tong, X. Deng, H. Chen, J. Mei, H. Liu, Ql-heft: a novel machine learning scheduling scheme base on cloud computing environment, Neural Comput. Appl. (2018) 1–18.
[14] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.
[15] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, M. Guizani, Security in mobile edge caching with reinforcement learning, IEEE Wireless Commun. 25 (3) (2018) 116–122.
[16] H. Wang, M. Gu, Q. Yu, Y. Tao, J. Li, H. Fei, J. Yan, W. Zhao, T. Hong, Adaptive and large-scale service composition based on deep reinforcement learning, Knowl.-Based Syst. 180 (2019) 75–90.
[17] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[18] M. Tiwary, D. Puthal, K.S. Sahoo, B. Sahoo, L.T. Yang, Response time optimization for cloudlets in mobile edge computing, J. Parallel Distrib. Comput. 119 (2018) 81–91.
[19] B.P. Rimal, D.P. Van, M. Maier, Mobile edge computing empowered fiber-wireless access networks in the 5g era, IEEE Commun. Mag. 55 (2) (2017) 192–200.
[20] I.A. Elgendy, W. Zhang, Y.-C. Tian, K. Li, Resource allocation and computation offloading with data security for mobile edge computing, Fut. Gen. Comput. Syst. 100 (2019) 531–541.
[21] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81.
[22] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Select. Areas Commun. 34 (12) (2016) 3590–3605.
[23] C. Liu, K. Li, J. Liang, K. Li, Cooper-sched: a cooperative scheduling framework for mobile edge computing with expected deadline guarantee, IEEE Trans. Parallel Distrib. Syst. (2019), 1–1.
[24] A. Samanta, Z. Chang, Z. Han, Latency-oblivious distributed task scheduling for mobile edge computing, in: 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, 2018, pp. 1–7..
[25] W. Chen, C.-T. Lea, K. Li, Dynamic resource allocation in ad-hoc mobile cloud computing, in: 2017 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2017, pp. 1–6..
[26] F. Wang, J. Xu, X. Wang, S. Cui, Joint offloading and computing optimization in wireless powered mobile-edge computing systems, IEEE Trans. Wireless Commun. 17 (3) (2017) 1784–1797.
[27] J. Hu, C. Liu, K. Li, K. Li, Game-based multi-md with qos computation offloading for mobile edge computing of limited computation capacity, in: IFIP International Conference on Network and Parallel Computing, Springer, 2019, pp. 16–27.
[28] X. Cao, F. Wang, J. Xu, R. Zhang, S. Cui, Joint computation and communication cooperation for energy-efficient mobile edge computing, IEEE Internet Things J. 6 (3) (2019) 4188–4200.
[29] M. Chen, Y. Hao, Task offloading for mobile edge computing in software defined ultra-dense network, IEEE J. Select. Areas Commun. 36 (3) (2018) 587–597.
[30] Y. Cao, H. Chen, J. Jiang, F. Hu, Tasrd: task scheduling relying on resource and dependency in mobile edge computing, in: 2018 IEEE International Conference on Progress in Informatics and Computing (PIC), IEEE, 2018, pp. 287–295.
[31] J. Zhang, X. Hu, Z. Ning, E.C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, B. Hu, Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks, IEEE Internet Things J. 5 (4) (2017) 2633–2645.
[32] T.Q. Dinh, J. Tang, Q.D. La, T.Q. Quek, Offloading in mobile edge computing: task allocation and computational frequency scaling, IEEE Trans. Commun. 65 (8) (2017) 3571–3584.
[33] Y. Mao, J. Zhang, S. Song, K.B. Letaief, Power-delay tradeoff in multi-user mobile-edge computing systems, in: 2016 IEEE Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–6..
[34] Y. Wang, K. Wang, H. Huang, T. Miyazaki, S. Guo, Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications, IEEE Trans. Ind. Inf. 15 (2) (2018) 976–986.
[35] Z. Ning, P. Dong, X. Wang, J.J. Rodrigues, F. Xia, Deep reinforcement learning for vehicular edge computing: An intelligent offloading system, ACM Trans. Intell. Syst. Technol. (TIST) 10 (6) (2019) 60..
[36] L. Huang, X. Feng, L. Qian, Y. Wu, Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing, in: International Conference on Machine Learning and Intelligent Communications, Springer, 2018, pp. 33–42..
[37] J. de Lope, D. Maravall, Y. Quiñonez, Self-organizing techniques to improve the decentralized multi-task distribution in multi-robot systems, Neurocomputing 163 (2015) 47–55.
[38] Y. Mao, J. Zhang, K.B. Letaief, Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems, in: 2017 IEEE wireless communications and networking conference (WCNC), IEEE, 2017, pp. 1–6..
[39] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[40] J. Chen, K. Li, K. Bilal, K. Li, S.Y. Philip, et al, A bi-layered parallel training architecture for large-scale convolutional neural networks, IEEE Trans. Parallel Distrib. Syst. 30 (5) (2018) 965–976.
[41] N. Kan, J. Zou, K. Tang, C. Li, N. Liu, H. Xiong, Deep reinforcement learning-based rate adaptation for adaptive 360-degree video streaming, in: ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 4030–4034.
[42] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep q-learning, Inf. Sci. (2019) 1–22.
[43] O. Anschel, N. Baram, N. Shimkin, Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, in: Proceedings of the 34th International Conference on Machine Learning, vol. 70, JMLR. org, 2017, pp. 176–185.
[44] A. Delgado, I. Romero, Environmental conflict analysis using an integrated grey clustering and entropy-weight method: a case study of a mining project in peru, Environ. Model. Software 77 (2016) 108–121.