sensors

Article

Distributed Learning Based Joint Communication and Computation Strategy of IoT Devices in Smart Cities

Tianyi Liu [1], Ruyu Luo [2], Fangmin Xu [3,*], Chaoqiong Fan [3] and Chenglin Zhao [3]

1 International School, Beijing University of Posts and Telecommunications, Beijing 100876, China; icebound@bupt.edu.cn
2 School of Information and Telecommunication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China; luory@bupt.edu.cn
3 Key Laboratory of Universal Wireless Communications, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China; june_fcq0618@bupt.edu.cn (C.F.); clzhao@bupt.edu.cn (C.Z.)
* Correspondence: xufm@bupt.edu.cn

Received: 30 December 2019; Accepted: 10 February 2020; Published: 12 February 2020

Abstract: With the development of global urbanization, the Internet of Things (IoT) and smart cities are becoming hot research topics. As an emerging model, edge computing can play an important role in smart cities because of its low latency and good performance. IoT devices can reduce time consumption with the help of a mobile edge computing (MEC) server. However, if too many IoT devices simultaneously choose to offload the computation tasks to the MEC server via the limited wireless channel, it may lead to the channel congestion, thus increasing time overhead. Facing a large number of IoT devices in smart cities, the centralized resource allocation algorithm needs a lot of signaling exchange, resulting in low efficiency. To solve the problem, this paper studies the joint policy of communication and computing of IoT devices in edge computing through game theory, and proposes distributed Q-learning algorithms with two learning policies. Simulation results show that the algorithm can converge quickly with a balanced solution.

Keywords: smart city; Internet of Things; mobile edge computing; potential game; Q-learning

1. Introduction

With the increasing number of cities and urban population, there has been an increasing interest in the smart city. The concept of smart city emphasizes a solution that takes into account the sustainability of the city in all regions and project implementation phases [1]. With making full use of a large number of interconnected devices, smart cities have been attached to the Internet of Things (IoT) technology. In the smart city, the automation strategy based on massive IoT devices deployment to gather Big Data to get insights into city behavior to improve its services [2]. In addition, the term Internet of Things represents the ability of intelligence devices to sense, collect, share data across the Internet or a local network, and then act according to the received information [3]. With the rapid increase of smart connected objects (such as personal devices, sensors, actuators) in the smart city, different simultaneous interpreting of delay is required by different time critical IoT applications. For example, early warning systems or live event broadcasting, present particular challenges [4–6]. In addition, many IoT devices require enhanced computing performance, such as real-time camera identification. To sum up, how to improve the quality of service (QoS) to ensure the normal operation of smart city has become an important issue.

In order to meet this challenge, mobile edge computing (MEC) [7] can efficiently improve the QoS for applications that require intensive computations and low latency, which have been raised to deploy computing resources closer to end devices in the smart city [8]. Once the IoT devices have started being put online, a new step in the evolution of mobile networks was taken through the addition of edge and fog computing, where small nodes at the edge of the network take up some of the load on the cloud backend [6]. In addition, when new IoT devices are connected due to urban construction, MEC is very convenient and flexible because it adopts wireless access. As shown in Figure 1, various IoT devices in the smart city can connect to the MEC servers. Through the MEC server, the IoT devices can perform the computation offloading, thus reducing the computation time consumption and resulting in lower latency.

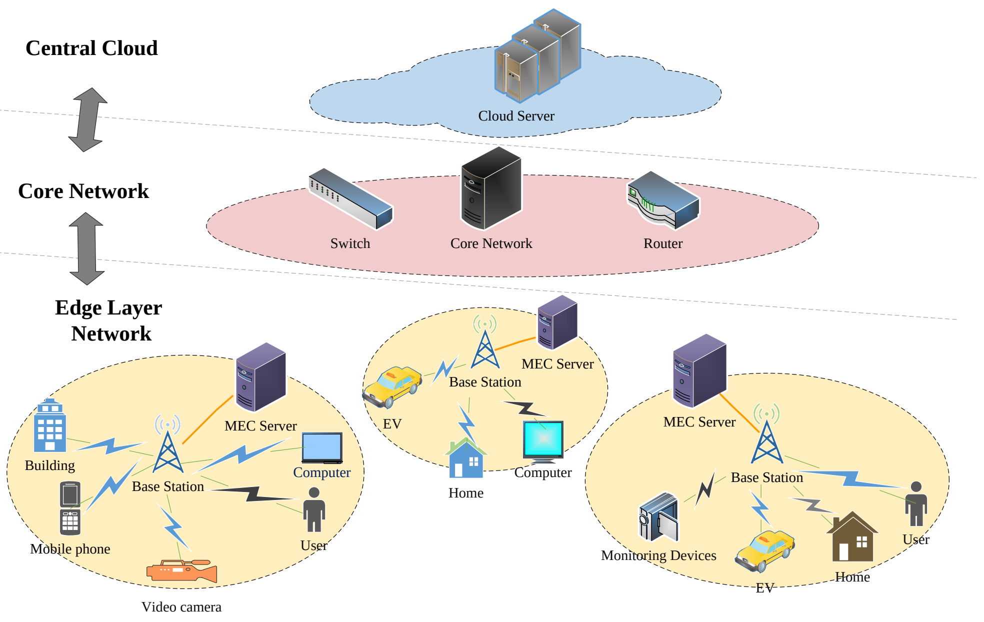

Figure 1. Applications of mobile edge computing in the smart city.

It is quite good to use MEC in the smart city, but resource allocation in the smart city is still a problem to be solved. First of all, if a large number of devices simultaneously do the computation offloading to the MEC server, due to the limited channel resources, the wireless network may become congested, thus increasing the time consumption. Second, IoT devices in smart cities are different from traditional sensors, many of which have some computing capability (such as Raspberry Pi [9]). If IoT devices adopt edge computing completely, this part of the computing resources will be wasted. Third, the smart city is generally composed of many parts, including many different services which has different requirements. A reasonable communications and computing strategy should ensure that as many IoT devices as possible are functioning properly, rather than having a small number of devices occupy all resources so that some parts of a smart city cannot work due to lack of resources. Moreover, the centralized resource allocation algorithm with a large amount of information collection and signaling exchange may not be suitable for smart city scenarios.

For the above problems and challenges, this paper focuses on how to perfectly do the resources (including wireless spectrum resources and computing resources) allocated in the smart city, mainly aiming at making as many IoT devices associated with one MEC server as possible accomplish the tasks in a low latency. We use distributed Q-learning to allocate resources and combine game theory to ensure the balance of solutions. In addition, we also creatively used two different learning policies. The main contributions of our propose scheme are as follows:

- Based on the system model of smart city and MEC, we formulate a computation offloading game with IoT devices in the smart city, and prove that the game has at least one Nash equilibrium point.

- Combined with game theory and system model, we proposed two distributed Q-learning algorithms with different learning policies to get the joint communication and computation strategies for each IoT devices in the smart city. The distributed Q-learning requires only a small amount of signaling exchange and has a good convergence performance, which is very suitable for smart cities. The combination of Q-learning and MEC can effectively improve the QoS of the smart city system.

The rest of this paper is organized as follows. In section 2, we review the related work. In Section 3, we introduce our system model and describe our problem. In Section 4, we design a noncooperative game model for computation offloading, which can prove the existence of Nash equilibrium. In Section 5, we describe two stateless distributed Q-learning algorithms with different learning policies for edge computation offloading game. In Section 6, simulation results are provided, showing that the distributed Q-learning algorithm has good performance. The conclusions and future work are given in Section 7.

2. Related Work

In recent years, in order to make full use of the computing and spectrum resources, many studies of communication and computation strategies have proposed. Some methods are based on cloud or MEC, and a few are specific to the smart city.

There are different methods meeting the challenge of communication and computation in different application scenarios for different purposes. Some of those strategies are developed based on cloud computing. Štefanič has introduced a general SWITCH architecture with its subsystems, TOSCA orchestration standard, and software engineering workflow in SWITCH to address entire life cycle of time-critical cloud applications [4]. Zeng has presented an architecture of multiple cloud service providers (CSPs) or "Cloud-of-Clouds" to provide services to the continuous writing applications (CWA) by using a novel resource scheduling algorithm to minimize the cost of entire systems [10]. Furthermore, it is very popular to use mobile edge computing (MEC) to slove communication and computation problems. Some have proposed the centralized algorithm, which needs to upload all information to a central node. Huang has proposed a control scheme for offloading vehicular communication traffic in the cellular network to vehicle-to-vehicle (V2V) paths using software-defined network (SDN) inside the MEC architecture [11]. Berno has considered a model for the allocation of processing tasks in MEC, and its allocation problem is formulated as a centralized (offline) optimization program with delay constraints (deadlines) [12]. Meanwhile, the distributed algorithm is also worthy of attention. T. Q. Dinh proposed a learning method for computation offloading in MEC, which is based on non-cooperative game and it focuses on maximizing CPU cycles [13]. Chen designed a distributed computation offloading algorithm that can achieve a Nash equilibrium for mobile-edge cloud computing [14]. Ranadheera summarized the previous work and compared the characteristics and advantages of different distributed algorithms in MEC [15]. Most of these distributed algorithms are based on game theory, giving us a lot of inspiration.

Considering the guarantees for low-latency applications in smart cities and the limited spectrum-computation and sub-channel resource, traditional cloud computing faces great challenges [16–18]. Thus, people have proposed several ways to complete the shortcomings of cloud computing in the smart city. Some solutions are still based on cloud computing. Kakderi has focused on the storm cloud paradigm as a solution to deploy a portfolio of smart cities applications on a single cloud-based platform and migrating existing applications to the cloud environment using the platform and its accompanied tools [5]. Ciobanu aimed at solving the problem of offloading data and computing from mobile devices to cloud, to fog nodes, or to other nearby mobile devices. The novelty of the proposal is to add a layer composed exclusively of mobile devices that collaborate in an opportunistic fashion [6]. There are also schemes based on data fusion. Facing the large amount of data generated by IoT devices, Esposito proposed an event-based data fusion approach, which effectively utilizes limited resources [19]. Moreover, there are also some studies based on MEC. Sapienza proposed a solution to detect abnormal or critical events such as terrorist threats, natural, and man-made disasters by using MECs. The proposed solution allows cooperation among the Base Transceiver Stations to rapidly notify the users which are close to the critical area[20]. Zhao proposed that edge servers could be set up in smart cities to improve QoS, and provided two methods of edge resource allocation, Enumeration-Based algorithm, and Clustering-Based algorithm [21]. Deng proposed four methods to schedule tasks for computation-intensive and time-sensitive smart city applications with the assistance of IoT based on multi-server mobile edge computing [22]. However, for the smart city scenario, studies based on learning and game theory are rare.

To sum up, there have been many previous attempts to solve resource allocation problems in smart cities, but few are based on game theory and learning algorithms. Therefore, we improved the Q-learning algorithm based on MEC, and proposed two methods with different learning policies for smart city.

3. System Model

Smart city is a huge system. In the smart city, a large number of services running over massive end IoT devices as well as applications hosted in remote servers. These IoT devices come in a wide variety of categories and services. It is very difficult to study the model of the whole smart city. Thus, we only consider one block in the smart city. In this paper, the uplink of wireless network integrated with multiple IoT devices is mainly considered. Furthermore, we aim at making as many IoT devices associated with one MEC server as possible meet the threshold and minimizing the total time delay. Thus, in the following article, in a block of the smart city, it is assumed that IoT devices in this block are associated with one MEC server. The set of IoT devices is presented by $\mathcal{M} = \{1, 2, \cdots, M\}$. Moreover, the set of IoT devices associated with the MEC server shares $\mathcal{N} = \{1, 2, \cdots, N\}$ orthogonal sub-channels. In the following article, we will show that the communication and computation model for the IoT devices in the smart city, and then describe our main problem.

3.1. Communication Model

First, we introduce the communication model for wireless access of the IoT devices. It is assumed that the distance between the IoT device m and its MEC server is d_m. For the links between IoT devices and MEC server, the sub-channel gain from the MEC server to IoT device m is defined as h_m. In addition, in our model, we assume that the sub-channel gains h_m are composed of sub-channel path loss PL and shadow fading. We assume that, for all IoT devices, the log-normal shadowing fading g_m is independent with zero mean and 8 dB standard derivation. $PL(d_m)$ represents the path loss at a reference distance d_m. Thus, the sub-channel gain can be computed as:

$$h_m = PL(d_m)g_m \tag{1}$$

In addition to meet the goal, optimal sub-channel allocation is important to support the IoT devices so that we need to consider the interference between the IoT devices. It is assumed that the transmit power of the IoT device m associated with the MEC server is denoted by P_m. I_m^n is the interference between the IoT device m and other IoT devices which share the same sub-channel n. It can be expressed as:

$$I_m^n = \sum_{i \in \mathcal{M} \setminus \{m\}} \zeta_i^n P_i h_i \tag{2}$$

In particular, I_m^0 is equal to zero for all devices. In addition, ζ_i^m is the sub-channel assignment indicator, and:

$$\zeta_m^n = \{0, 1\} \qquad \forall m, n \tag{3}$$

$$\sum_{n=1}^{N} \zeta_m^n = 1 \qquad \forall m \tag{4}$$

where $\zeta_m^n = 1$ indicates that the device m use the sub-channel n . According to Shannon formula, the instantaneous rate of the IoT device m on sub-channel n can be formulated as

$$r_m^n = W_n \log_2 (1 + \frac{P_m h_m}{\delta^2 + I_m^n}) \tag{5}$$

In the above formulate, δ^2 is the thermal noise, W_n is the sub-channel bandwidth.

3.2. Computation Model

Then, we introduce the computation model. In the smart city system, it is assumed that each IoT device will handle a specific type of task in a period time. The set of computation tasks is $\mathcal{J}$. For the IoT device m, it has a kind of computation task $\mathcal{J}_m \in \mathcal{J}$ and $\mathcal{J}_m = (D_m, C_m, TH_m)$, where D_m is the amount of input data for the task, including all parameters and codes. C_m is the number of CPU cycles required to accomplish the computation task. In addition, TH_m is a time threshold for task $\mathcal{J}_m$. If $T_m \leq TH_m$, task $\mathcal{J}_m$ is considered to be successfully completed. For convenience of representation, we refer to the IoT device m in this case as the device that meets the threshold. Otherwise, task $\mathcal{J}_m$ is overtime. The device m will accomplish the task locally or in the edge. Both approaches are discussed as follows.

3.2.1. Local Computing

For the local computing approach, IoT device m will execute task $\mathcal{J}_m$ locally by its own computation ability. Different devices have different computation capabilities in a smart city system. For local computing, let f_m^l be the computation capability (i.e., CPU cycles per second) of the IoT device m. It should be noted that, when designing a smart city system, the designer should consider the computation capability of the IoT devices. The devices need to have appropriate computing capability to meet their services and ensure that the system is not congested. In other words, it can be considered that the probability of queueing when the task is locally computed is very low. Therefore, in order to simplify the model, the queueing delay of tasks in local computation can be ignored. The computation execution time of the task $\mathcal{J}_m$ by local computing is then given as:

$$T_m^l = \frac{C_m}{f_m} \tag{6}$$

3.2.2. Edge Computing

For the edge computing, IoT device m will offload its task $\mathcal{J}_m$ to the nearby MEC server. Like some studies such as [23], for simplifying the problem, only one MEC server is deployed in one block of the city. In addition, our study focuses on one block. Thus, it is assumed that there is only one MEC server in the model with computation capability f^e. Therefore, the time for the MEC server to execute a task is:

$$t_m^e = \frac{C_m}{f^e} \tag{7}$$

When offloading a task, the device first selects a wireless sub-channel and uploads all data using 4G/5G or other wireless approaches to the MEC server. The time required for IoT device m uploading a task $\mathcal{J}_m$ with sub-channel n is

$$t_{m,n}^u = \frac{D_m}{r_n^m} \tag{8}$$

When the upload is complete, the task will be processed immediately. Based on the above formula, we can obtain the total expected time for IoT device m to complete a task $\mathcal{J}_m$ with sub-channel n using edge computing, which is expressed as follows:

$$T_m^e = t_{m,n}^u + t_m^e \tag{9}$$

Based on the above system model, we can estimate the time overhead for the IoT devices to accomplish tasks with edge computing and local computing. Similar to some studies such as [14], we only consider IoT devices uploading input data to the MEC server. The time required to transmit the output data is neglected in this model because the size of the output data in general is much smaller than the size of the input data, and the download rate is much faster compared to the upload rate. In addition, in the study of smart cities, wireless channel resources and computing resources are usually the scarcest and should be paid most attention to.

3.3. Problem Description

In the smart city system, MEC server and IoT devices generally have stable power supply, so we can neglect the power consumption of IoT devices and MEC server. It is more important to consider the time overhead of the task. By offloading tasks to the MEC server, IoT devices can effectively reduce time consumption. However, if too many IoT devices simultaneously choose to offload the computation tasks to the MEC server via the same wireless sub-channel, it may lead to severe sub-channel interference, resulting in long upload time. Therefore, finding the proper strategies adopted by each device is the key problem.

In the system, the processing latency of accomplishing tasks need to be minimized. For IoT device m, the expected time to finish task $\mathcal{J}_m$ is:

$$T_m = (1 - s_m) * T_m^l + s_m * T_m^e \tag{10}$$

where s_m is the computation state of device m. If $s_m = 1$, the device chooses edge computing. Otherwise, the device finishes its task locally. We want to accomplish tasks successfully as many as possible and spend as little time overhead as possible on each task. Therefore, for one device m, our study target is to find a proper computing strategy (including computing state and wireless sub-channel selection) to minimize T_m for meeting the threshold TH_m. We consider that every IoT device is selfish and want to reduce their task processing latency. Based on this condition, a game-theory based model will be introduced to formulate and solve the problem.

4. Non-Cooperative Game Model for Computation Offloading

According to the problem description in Section 3, it is shown that the computation decisions among IoT devices are coupled. The time overhead of a device may be influenced by the decision of other devices. However, in the distributed decision-making system, each user's decision is independent. In this case, similar to some studies [13,24], non-cooperative game theory can be applied to the problem. In the non-cooperative game, each IoT device is selfish and they want to minimize their time overhead by making decision independently. In this section, we will show the game formulation and use the potential game to analyze the Nash equilibrium.

4.1. Game Formulation

Based on the system model in Section 3, for the IoT device m, we define its action as follows:

$$a_m = \begin{cases} \sum\limits_{i=1}^{N} (i\zeta_m^i) & s_m = 1 \\ 0 & s_m = 0 \end{cases} \tag{11}$$

The action $a_m \in \mathcal{A} = \{0, 1, ..., N\}$ represents the computational offloading decision for device m. It means that, when the device m choose edge computing, a_m is equal to the sub-channel index; otherwise, a_m is zero. Note that all the IoT devices have a same action space, so they can share the action set $\mathcal{A}$. For all the devices, the actions form an vector $\boldsymbol{a} = \{a_1, a_2, ...a_m\}$ representing the action vector (or strategy vector) of the devices. Let a special action vector $\boldsymbol{a}_{-m} = \{a_1, ..., a_{m-1}, a_{m+1}, ..., a_M\}$ be the actions decided by all other IoT devices except device m. Combining the other devices' actions a_{-m}, the device m can select a proper action a_m to minimize the time overhead. Then, we can formulate the problem as:

$$\mathcal{F} : \min_{a_m \in \mathcal{A}} U_m(a_m, a_{-m}) \tag{12}$$

where $U_m(a_m, a_{-m})$ is called utility function. It can be expressed as:

$$U_m(a_m, a_{-m}) = T_m = \begin{cases} T_m^e & a_m > 0 \\ T_m^l & a_m = 0 \end{cases} \tag{13}$$

Based on the above formula, we can formulate the problem as a strategic form game as the following definitions:

Definition 1. *Edge Computation Offloading Game (ECOG): We define a strategic form game $\Gamma = (\mathcal{M}, \mathcal{A}, \mathcal{U})$ as an edge computation offloading game, where*

$\mathcal{M}$ is the set of IoT devices in the smart city, which are the players in the game.

$\mathcal{A}$ is the action set shared by all the IoT devices.

$\mathcal{U} = \{U_m\}_{m \in \mathcal{M}}$ is the set of utility functions for all devices. U_m is defined in (13).

More definitions and proofs on game theory and strategic form game are given by [25]. Now, the definition of Nash equilibrium is as follows, which accounts for a steady state of an ECOG:

Definition 2. *Nash Equilibrium (NE): A strategy vector $\boldsymbol{a} = \{a_1^*, a_2^*, ...a_m^*\}$ is a Nash equilibrium of the ECOG if there is no IoT device can further reduce its utility by changing action unilaterally, i.e.:*

$$U_m(a_m^*, a_{-m}^*) \leq U_m(a_m, a_{-m}^*) \qquad \forall a_m \in \mathcal{A}, \forall m \in \mathcal{N} \tag{14}$$

Nash equilibrium is an important concept in game theory. For our problem, it has great meaning to the solution. If a device m chooses edge computing when the game is at NE, it must reduce the time overhead by computation offloading. Because if device m chooses edge computing and it spends more time to finish the task, it can change to local computing to decrease its utility, which is against the definition of a Nash equilibrium. In other words, the NE point must be a relatively good solution to the problem described in Section 3. Due to the concept of the Nash equilibrium, the IoT devices in NE can achieve a mutually satisfactory solution and no one has the intention to deviate the steady state.

Although NE is very important, we still don't know whether NE exists in ECOG. In the next part, a powerful tool, potential game will be introduced to prove the existence of Nash equilibrium.

4.2. Potential Game

Potential game is a strategic form game with a finite number of players. The definition of the ordinary potential game is as follows:

Definition 3. *Ordinary Potential Game (OPG): A strategic form game is called ordinary potential game (OPG) if it admits a potential function $\Phi_m(\boldsymbol{a})$ such that, for every $m \in \mathcal{M}$, $a_{-m} = \{a_1, ..., a_{m-1}, a_{m+1}, ..., a_M\}$ and $a_m, a_m' \in \mathcal{A}$, if*

$$U_m(a'_m, a_{-m}) < U_m(a_m, a_{-m}) \tag{15}$$

we have

$$\Phi_m(a'_m, a_{-m}) < \Phi_m(a_m, a_{-m}) \tag{16}$$

OPG has an appealing property that every ordinary potential game always has a pure strategy Nash equilibrium and the finite improvement property (FIP) [26]. We can use OPG to prove that the Nash equilibrium of ECOG exists, that is, proving ECOG is OPG. Before processing, we show the following result at first:

Lemma 1. *When the IoT device m chooses to select edge computing using wireless sub-channel n, its interference I_m^n must satisfied that $I_m^n < H_m$, with the threshold*

$$H_m = \frac{P_m h_m}{2^{\frac{D_m}{W_m(T_m^l - t_m^e)}} - 1} \tag{17}$$

Proof. Since device m chooses sub-channel n to do the tasks, the time overhead of edge computing must less than local computing, so we know that $T_m^e < T_m^l$. According to Section 3, this can be expressed as:

$$I_m^n < \frac{P_m h_m}{2^{\frac{D_m}{W_m(T_m^l - t_m^e)}} - 1} = H_m \tag{18}$$

Note that there is a very small probability that T_m^e is going to be equal to T_m^l. In that case, we assume that the device will choose local computing because it is more convenient for no uploading data. Then, Lemma 1 has been proved. □

Based on Lemma 1, we can determine the potential function of ECOG:

Theorem 1. *The edge computation offloading game is an ordinary potential game, which has at least one NE and the finite improvement property, with the potential function:*

$$\Phi_m(\boldsymbol{a}) = \sum_{i=1}^{M} P_i h_i I_i^{a_i} + 2\sum_{i=1}^{M}(1 - s_i) P_i h_i H_i \tag{19}$$

Proof. Firstly, supposing that a IoT device m changes its action from a_m to a'_m to reduce its utility, which is $U_m(a'_m, a_{-m}) < U_m(a_m, a_{-m})$. There are three cases to achieve this reduction: (1) $a_m, a'_m > 0$, (2) $a_m = 0, a'_m > 0$, and (3) $a_m > 0, a'_m = 0$.

For case (1), $a_m, a'_m > 0$ means that the device m chooses edge computing but changes the sub-channel to do the computation offloading. According to our hypothesis, the utility is decreased, such that $U_m(a'_m, a_{-m}) < U_m(a_m, a_{-m})$. Since device m just does the sub-channel changing, the processing time is no change. It implies that only an increase in the upload rate results in a reduction in time overhead:

$$W_m \log_2(1 + \frac{P_m h_m}{\delta^2 + I_m^{a_m}}) < W_m \log_2(1 + \frac{P_m h_m}{\delta^2 + I_m^{a'_m}}) \tag{20}$$

It is easy to obtain that:

$$I_m^{a'_m} < I_m^{a_m} \tag{21}$$

It means that the interference for device m is reduced, and devices that are computing locally are not affected. According to (19) and the definition of potential function, we know that:

$$\begin{aligned}
&\Phi_m(a'_m, a_{-m}) - \Phi_m(a_m, a_{-m}) = \sum_{i=1,\boldsymbol{a}=\{a'_m,a_{-m}\}}^{M} P_i h_i \sum_{j=1,j\neq i}^{M} \zeta_j^{a_i} P_j h_j - \sum_{i=1,\boldsymbol{a}=\{a_m,a_{-m}\}}^{M} P_i h_i \sum_{j=1,j\neq i}^{M} \zeta_j^{a_i} P_j h_j \\
&= P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a'_m} P_i h_i + \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j + \zeta_i^{a'_m} P_m h_m \right) \\
&\quad - P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a_m} P_i h_i - \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j + \zeta_i^{a_m} P_m h_m \right) \\
&= P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a'_m} P_i h_i - P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a_m} P_i h_i + P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a'_m} P_i h_i - P_m h_m \sum_{i=1,i\neq m}^{M} \zeta_i^{a_m} P_i h_i \\
&= 2 P_m h_m \left(I_m^{a'_m} - I_m^{a_m} \right)
\end{aligned} \tag{22}$$

Because of (21), the above expression indicates that $\Phi_m(a'_m, a_{-m}) < \Phi_m(a_m, a_{-m})$. Then, we use the same way to prove case (2). For case (2), we can firstly subtract the two potential functions:

$$\begin{aligned}
&\Phi_m(a'_m, a_{-m}) - \Phi_m(a_m, a_{-m}) = \sum_{i=1,\boldsymbol{a}=\{a'_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} - \sum_{i=1,\boldsymbol{a}=\{a_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} - 2 P_m h_m H_m \\
&= P_m h_m I_m^{a'_m} - 2 P_m h_m H_m + \sum_{i=1,i\neq m,\boldsymbol{a}=\{a'_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} - \sum_{i=1,i\neq m,\boldsymbol{a}=\{a_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} \\
&= 2 P_m h_m (I_m^{a'_m} - H_m)
\end{aligned} \tag{23}$$

where H'_i is the threshold of $\boldsymbol{a} = \{a_m, a_{-m}\}$. Since $U_m(a'_m, a_{-m}) < U_m(a_m, a_{-m})$ and $a_m = 0, a'_m > 0$, it means that the device m changes to edge computing for reducing time overhead. According to Lemma 1, the interference of device m must satisfy a threshold H_m that is $I_m^{a'_m} < H_m$. Thus, case (2) is proved. For case (3), it is very similar to case (2). We use the same method in case (2) to do the subtraction:

$$\begin{aligned}
&\Phi_m(a'_m, a_{-m}) - \Phi_m(a_m, a_{-m}) = \sum_{i=1,\boldsymbol{a}=\{a'_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} + 2 P_m h_m H_m - \sum_{i=1,\boldsymbol{a}=\{a_m,a_{-m}\}}^{M} P_i h_i I_i^{a_i} \\
&= 2 P_m h_m H_m - P_m h_m I_m^{a_m} + \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j \right) - \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j + \zeta_i^{a_m} P_m h_m \right) \\
&= P_m h_m I_m^{a'_m} - 2 P_m h_m H_m + \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j + \zeta_i^{a'_m} P_m h_m \right) - \sum_{i=1,i\neq m}^{M} P_i h_i \left(\sum_{j=1,j\neq i,j\neq m}^{M} \zeta_j^{a_i} P_j h_j \right) \\
&= 2 P_m h_m (H_m - I_m^{a_m})
\end{aligned} \tag{24}$$

Similar to case (2), when the device change to local computing, we have $I_m^{a_m} > H_m$. Then, case (3) is proved. Combining results in the three cases, we can conclude that the edge computation offloading game is an ordinary potential game with potential function as given in (19). Then, Theorem 1 is proved. $\square$

In this section, the definition of the ECOG is given. In addition, the properties of ECOG have been determined. Then, we will use the properties of the potential game to find the solution by Q-learning.

5. Distributed Q-learning Algorithm for Computation Offloading

As a popular algorithm in a channel allocation problem, Q-learning is often used in combination with game theory. In the smart city, each device's perception of the environment is very limited, and it is difficult to represent the state of the system. Therefore, we apply a distributed and stateless

Q-learning to solve the computation offloading problem. In this part, we will introduce a distributed reinforcement learning algorithm called stateless Q-learning and give two learning policies.

5.1. Stateless Q-Learning

In the smart city, it is so difficult for IoT devices to observe and record the state (such as other devices' addictions) for the system. They can only sample and estimate their environment. Due to the small amount of information available to the device, a simple stateless variation of Q-learning, as formulated in [27], is used for our problem. In a smart city system with a lot of IoT devices, this reinforce learning algorithm is distributed and all IoT devices do not need to exchange information with others, resulting high efficiency.

In the Q-learning procedure, we formulate the Q-value for device m at time k as below:

$$\forall i \in \mathcal{A}, \quad Q_m^i(k) = \begin{cases} (1-\alpha_m(k))Q_m^i(k-1) + \alpha_m(k) r_m^i(k) & i = a_m(k) \\ Q_m^i(k-1) & otherwise \end{cases} \tag{25}$$

where $\alpha_m(k)$ is the learning rate of device m at time k, and $a_m(k)$ is the action taken by device m at time k. $r_m^i(k)$ is the reward that the device m chooses to take action i in time k. Based on Section 3, we define $r_m^i(k)$ as:

$$r_m^i(k) = TH_m - T_m(k) \tag{26}$$

$T_m(k)$ is the expected time for device m to finish task at time k. It means that, if the task is overtime, the IoT device will get a negative reward. Otherwise, the less time consumed, the greater reward for the devices. For each IoT device, they run the same distributed learning algorithm without synchronization. When distributed Q-learning start, each IoT device selects an action following to a policy in each iteration, then gets a reward to update the Q-table, and finally converges to a solution. Next, two Q-learning algorithms of different learning policies will be introduced, and the analysis of convergence (to NE) will be given.

5.2. Distributed Q-Learning with ϵ-Greedy Learning Policy

The ϵ-greedy learning policy is common in Q-learning. In our ECOG, the actions of devices in the Q-learning will converge to NE by taking ϵ-greedy policy. To show this policy, we need to define the ϵ_k as:

$$\epsilon_k = \epsilon_0 k^{-1/M}, \epsilon_0 \in (0,1) \tag{27}$$

In ϵ-greedy selection policy, an IoT device m selects an action with maximum Q-value at time k with probability $(1-\epsilon_k)$ and chooses other actions randomly with probability ϵ_k. It can be expressed as:

$$a_m(k) = \begin{cases} \arg\max_i \left\{Q_m^i(k-1)\right\} & w.p.(1-\epsilon_k) \\ \text{randomly selected from}[0,N] & w.p.\epsilon_k \end{cases} \tag{28}$$

In addition, in order for the algorithm to converge to NE, the learning rate must follow the condition that:

$$\sum_{k=1}^{\infty} \alpha_m(k) = \infty \quad and \quad \sum_{k=1}^{\infty} (\alpha_m(k))^2 < \infty \tag{29}$$

Thus, we can achieve that by define the $\alpha_m(k)$ as:

$$\alpha_m(k) = \left[\beta + \Delta_m^k(a_m(k))\right]^{-\rho} \tag{30}$$

where $\Delta_m(a_m(k))$ is the number of times the action $a_m(k)$ has been selected by device m up to time k. $\rho \in (0.5, 1]$ is the rate parameter. When ρ increases, the learning rate decreases faster, so the algorithm converges faster. In addition, β is a positive constant associated with the initial value of the learning rate. When β increases, the learning rate increases and the algorithm will converge faster. Note that the other learning rate that follows the condition (29) is feasible for the distributed stateless Q-learning. We just take a special case based on our experiences. In addition, since it is difficult to judge whether the algorithm converges, we define a sufficiently large number of iterations k^*. The algorithm will stop after k^* iterations. After defining the relevant formulas, the summarization of the distributed Q-learning with ϵ-greedy learning policy is shown in Algorithm 1. Because the algorithm is distributed, there is no need to be synchronized. This reduces the signaling transmission in the iterative process and greatly improves the efficiency of the algorithm. For the result, if each IoT device in the smart city takes the Q-learning algorithm with an ϵ-greedy learning policy, the actions will converge to an NE point of an ECOG. Now, we will prove it.

Algorithm 1 Distributed Q-learning for device m with ϵ-greedy learning policy.

```
for $i = 0$ to $N$ do
    $Q^i_m(0) = 0$
end for
for $k = 1$ to $k^*$ do
    $\epsilon_k = \epsilon_0 k^{-1/N}$
    Follow the $\epsilon$-greedy policy (28) to get the next action $a_m(k)$.
    Based on $a_m(k)$, change the computing state and sub-channel.
    Observe sub-channel interference and calculate the reward $r_m^{a_m(k)}(k)$ according to (26).
    for $i = 0$ to $N$ do
        Calculate the $Q^i_m(k)$ according to (25) with the learning rate formulated at (30)
    end for
end for
```

Theorem 2. *For an ECOG, consider a distributed stateless Q-learning with an ϵ-greedy learning policy. Each time, the device selects the action according to (28) and updates the q-table according to (25) and (30). Then, for a sufficiently large number of iterations, k, the actions of all devices will converge to NE with probability 1.*

Proof. At the beginning, we define that a strategic form game is a weakly acyclic game (WAG) if, for any joint strategy, there exists a finite improvement path that starts at it. It is clearly that every OPG is weakly acyclic game [28]. Thus, ECOG is also a kind of WAG. Then, the result can be proved in three parts. First, we need to show that the process is weakly ergodic. Second, prove that the process is strongly ergodic. Finally, using the nature of WAG, it is proved that actions can only converge to NE. The proof is so complicated that it is omitted here. See [29] for details of the proof. □

Theorem 2 shows the stability of Q-learning. Regardless of the environment, stateless Q-learning with ϵ-greedy learning policy will certainly make ECOG converge to NE and give a relatively good solution. This is of great help to the problem of computing unloading in the smart city.

5.3. Distributed Q-Learning with Boltzmann Learning Policy

Compared with ϵ-greedy learning policy, Boltzmann learning policy is more sensitive to Q-values. Devices have a higher probability of choosing actions that have a higher Q-value. This policy, which has an obvious preference for the selection of Q-values, makes the distributed Q-learning with this learning policy have excellent convergence speed. In Boltzmann learning policy, each device needs to maintain an action selection vector $\boldsymbol{p}_m(k)$ as shown below:

$$p_m^i(k) = \frac{e^{Q_m^i(k)/\rho_m(k)}}{\sum\limits_{j=0}^{N} e^{Q_m^j(k)/\rho_m(k)}}, \quad \forall i \in \mathcal{A} \tag{31}$$

where $\rho_m(k)$ is called virtual temperature depended on the cooling function [30]. In our algorithm, it is expressed as:

$$\rho_m(k) = \frac{\rho_0}{(\log_2(k+1))^{\min\{2.5,(2M+k-1)/(2M)\}}} \tag{32}$$

This cooling function is based on experience and simulation, where ρ_0 is a positive constant called initial temperature. The temperature decreases slowly at first and then rapidly, resulting in the convergence of the actions. Specifically, in the process of algorithm execution, IoT devices need to choose actions according to its action selection vector which is determined by (31). At the beginning, the virtual temperature is high so that the selection probability of each action is close, which leads to more explorations of actions. This means more accurate estimates of the environment. Then, as the temperature gradually decreases, the device will be inclined to choose an action with a higher Q-value. In addition, the learning rate for this algorithm is a positive constant less than 1, that is,

$$\alpha_m(k) = A0, \quad 0 < A0 < 1 \tag{33}$$

Based on the above description, pseudo code of distributed Q-learning with Boltzmann learning policy is shown as Algorithm 2.

Algorithm 2 Distributed Q-learning for device m with Boltzmann learning policy.

```
for $i = 0$ to $N$ do
    $Q_m^i(0) = 0$
    $p_m^i(0) = 1/(N+1)$
end for
for $k = 1$ to $k^*$ do
    if max $p_m^i(k) \geq 0.99$ then
        exit.
    end if
    Choose the action $a_m(k)$ according to its current action selection probability vector $\boldsymbol{p}_m(k)$.
    Based on $a_m(k)$, change the computing state and sub-channel.
    Observe sub-channel interference and calculate the reward $r_m^{a_m(k)}(k)$ according to (26).
    for $i = 0$ to $N$ do
        Calculate the $Q_m^i(k)$ according to (25) with the learning rate formulated at (33)
        Update the action selection probability vector according to (31).
    end for
end for
```

Since the virtual temperature will decrease, the probability of choosing the action with the largest Q-value will increase and the action of the device must be convergent. In order to prevent a dead cycle, the algorithm will stop when the k reaches the maximum number of iterations k^*. In our simulation, Q-learning with Boltzmann learning policy generally converges very quickly with a good result.

6. Simulation Result

In this section, some numerical simulation results will be provided to evaluate the performance and convergence of the distributed Q-learning algorithm. In our simulation, the MEC server will be located in the central of an area. There are M IoT devices randomly located around the MEC server. Without loss of generality, it is assumed that $n = 5$ wireless sub-channels are available in this area. The sub-channel bandwidth is 0.8,1,2,5,10 Mhz, respectively. For the IoT devices, we assume that different devices have different computation capabilities. The parameters of IoT devices and MEC server are shown in Table 1.

Table 1. The parameters of Internet of Things (IoT) devices and mobile edge computing (MEC) server

Parameter	Description	Value	Unit
PL	Pathloss	$35.3 + 37.6 * log10(d)$	dB
δ^2	Thermal noise	-179.0	dBm/Hz
d	Distance between devices and server	$[200, 800]$	meter
f_i	Computation capability of devices	$\{2000, 3300\}$	cycle/s
f^e	Computation capability of MEC server	7000	cycle/s
P_i	Transmit power of IoT devices	20	dBm

To get closer to a real smart city, we designed five different computing tasks. Each task has its own characteristics, as shown in Table 2.

Table 2. The tasks in the smart city

Task Type i	D_i	C_i	TH_i
0	7000 Kb	20,000 cycles	7 s
1	4000 Kb	82,000 cycles	22 s
2	2200 Kb	150,000 cycles	45 s
3	10,000 Kb	100,000 cycles	28 s
4	5000 Kb	10,000 cycles	3 s

Since there are so many applications and services in the smart city, it is considered that each task is a real task in the smart city. For example, task 3 has a large size input and requires a lot of computation, which is very similar to the human activity recognition task. Task 4 only requires a small amount of calculation to complete, and we can think of it as a simple task of uploading and synchronizing traffic information.

In addition, for a Q-learning algorithm with ϵ-greedy learning policy, ϵ_k is equal to 0.125 and its learning rate is $\alpha_m(k) = \left[0.5 + \Delta_m^k(a_m(k))\right]^{-0.6}$ for all $m \in \mathcal{M}$. In addition, for Q-learning with Boltzmann learning policy, it has an initial virtual temperature $\rho_0 = 3$ and a fixed learning rate $\alpha_m(k) = 0.3$. To evaluate the performance of the two algorithms, we introduce another distributed computation offloading algorithm (DCOA) [14] for comparison. The DCOA is based on the FIP of the potential game. In each iteration, the MEC server will send the decisions of all devices to each device, and each device will calculate the best decision it can get if the other devices do not change their decisions. MEC server will then randomly select only one device to change its decision. When all the devices can not find the next best decision, the DCOA stops. We also introduce all local computing for comparison, that is, all devices only complete tasks locally. Next, we will show our simulation results and give the analysis.

Firstly, to evaluate the performance of the result, for a certain M, we randomly generated 1000 sets of test parameters, and run the four methods using these parameters. All algorithms are limited to 200 iterations. The results of M are 5, 10, 20, 30, 40, and 50 are shown in Figures 2 and 3. As shown in Figure 2, Q-learning algorithms perform well in terms of average time overhead. When $M = 5$, the average time overhead of strategies obtained by DCOA is the lowest. The results of Q-learning are not as good as DCOA, suggesting that DCOA has an advantage in scenarios with a small number of IoT devices. However, when $M > 5$, Q-learning can get the strategy of less time consumption. Moreover, with the increase of the number of access devices, by using the Q-learning algorithm, the delay does not increase significantly, and the system can still run smoothly without causing congestion. Overall, with MEC, the average time consumption of devices using Q-learning algorithm is 23% to 51% less than that of local computing. Then, we define that on-time rate is the ratio of the number of devices that meet the threshold. In addition, in Figure 3, the distributed Q-learning algorithm allows more devices to meet the threshold. When the number of devices is small, the Q-learning algorithm using Boltzmann learning policy has a good performance. However, with a large number of devices, Q-learning with ϵ-greedy learning policy works better. The on-time rate obtained by Q-learning is about 0.1 higher than that of DCOA, which indicates the good performance of Q-learning.

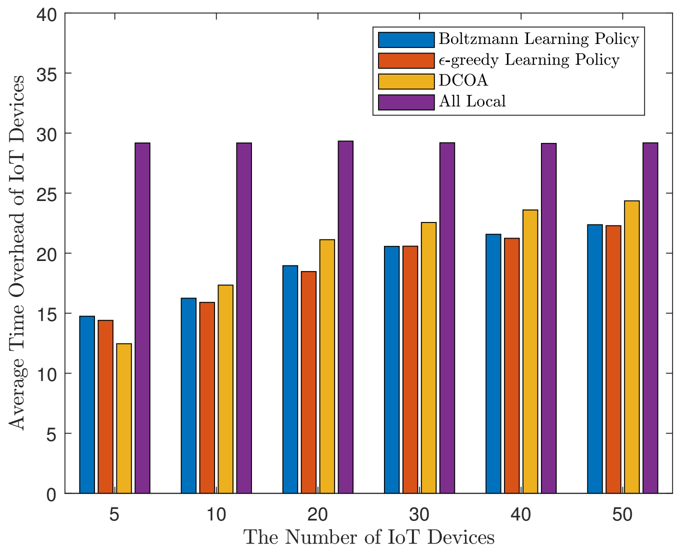

Figure 2. Average time overhead of the Internet of Things (IoT) devices.

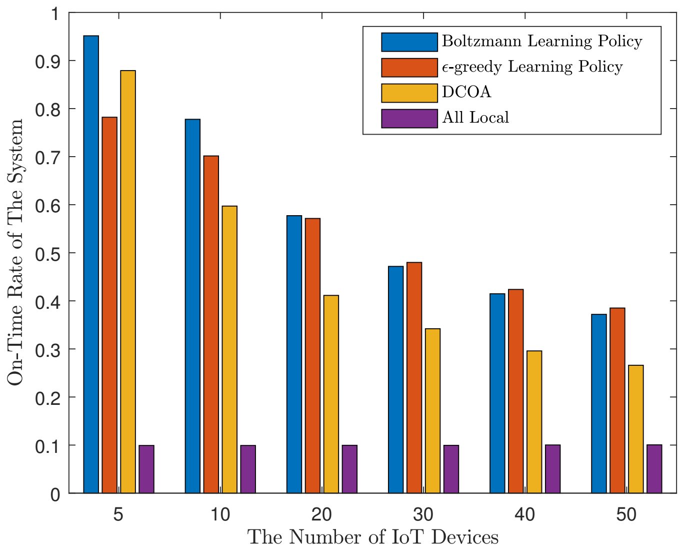

Figure 3. On-time rate of IoT devices.

Second, we will evaluate the convergence process of the algorithms. In this part, for $M = 10$ and $M = 30$, each algorithm will run 1000 times in one set of parameters. We forced all the algorithms to iterate 200 times. Then, take the average of the results after each iteration, and Figures 4 and 5 are obtained. These figures are the mean values of the results after many experiments, which can reflect the general convergence trend of the algorithms. However, they can not represent the exact speed of convergence. According to the figures, whether $M = 10$ or $M = 30$, both Q-learning algorithms rapidly converge to a solution. The process of convergence is the same as the description in Section 5. In Figures 4 and 5, Q-learning algorithms show a good performance in on-time rate and average time overhead. When $M = 10$, Q-learning using Boltzmann policy can achieve less time overhead and higher on-time rate. When $M = 30$, the performance of the two Q-learning is almost the same. In addition, in both scenarios, Q-learning performs better than DCOA. The solution obtained by Q-learning algorithm converges to NE, which is a stable state acceptable to all devices. In addition, the NE point obtained by Q-learning is better than that obtained by DCOA. Note that the time overhead and on-time rate of Q-learning fluctuated rather than changed monotonously. As Q-learning algorithm is based on learning, it will explore different solutions in the iterative process. This causes the curve to fluctuate. Finally, both Q-learning algorithms converged and stoped the fluctuation.

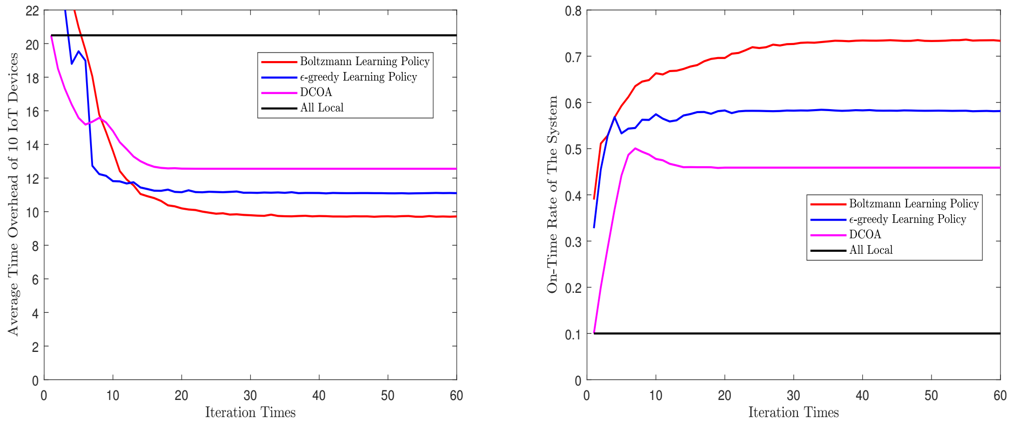

Figure 4. Convergence processes of different algorithms with 10 IoT devices.

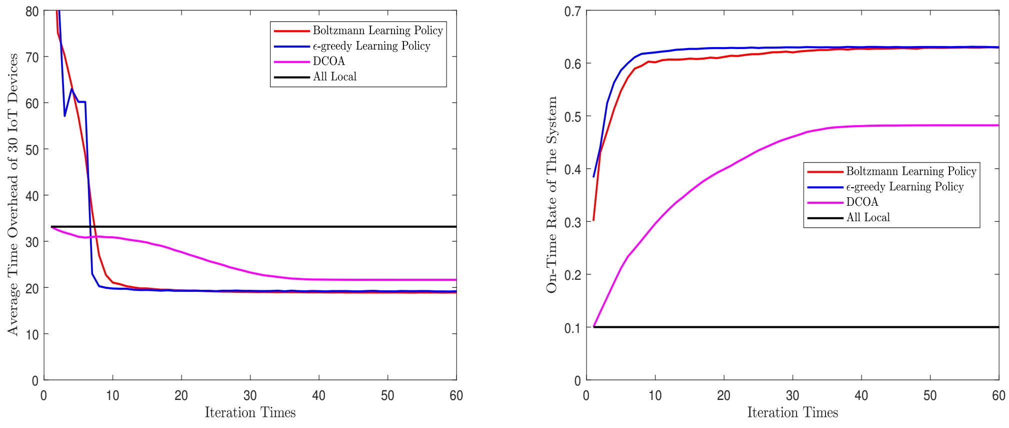

Figure 5. Convergence processes of different algorithms with 30 IoT devices.

In particular, compared to the Q-learning with ϵ-greedy policy, Q-learning with Boltzmann learning policy will stop immediately after reaching convergence, reducing meaningless iterations and improving algorithm efficiency. We can obtain the average iteration time required for convergence in the Q-learning with policy Boltzmann policy and DCOA. The results are shown as in Figure 6. In Figure 6, the average convergence times of Q-learning are related to the random exploration in each experiment, which can be used as a reference for the convergence speed in the actual smart city. It shows that the average number of times for convergence increases almost linearly as the number of IoT devices increases. In addition, the smaller ρ_0 is, the faster Q-learning converges. However, the convergence speed of Q-learning is significantly higher than that of DCOA because Q-learning needs exploration and learning to get a better result. However, DCOA needs to broadcast the decision of each IoT device, which requires more signaling exchange than Q-learning.

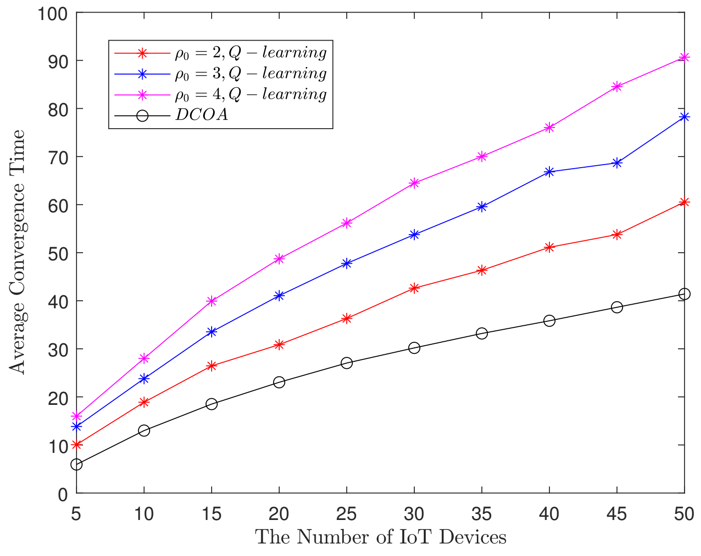

Figure 6. Convergence time of the Q-learning with Boltzmann learning policy and distributed computation offloading algorithm (DCOA).

7. Conclusions and Future Work

In the smart city, thousands of IoT devices simultaneously generate massive computing tasks to keep multiple services running in the city. Virtual reality, high-definition video, live event broadcasting, and other IoT applications put forward higher requirements on task processing time. Facing this challenge, this paper has focused on finding the proper computation offloading strategies with MEC server adopted by the IoT devices in the smart city. We apply a distributed reinforcement learning algorithm called stateless Q-learning and give two learning policies. This algorithm has many advantages in the smart city scenario. Firstly, it has been found that the distributed Q-learning algorithm has good performance in reducing time overhead and convergence. Compared with local computation, with the MEC server, the Q-learning algorithm can reduce the average time overhead by up to 51%. In addition, compared with DCOA, the on-time rate obtained by Q-learning is about 0.1 higher. In addition, according to the mathematical proof and simulation results, Q-learning can converge to NE, which means that the algorithm can produce a solution that is acceptable to all IoT devices. This avoids the congestion caused by uneven distribution of computing resources. Moreover, the distributed algorithm only needs a small amount of signaling exchange. Therefore, it is suitable for the smart city with lots of IoT devices.

In our future work, we may consider more general scenarios in the smart city. For example, combining with the queuing theory, we can create a computing model more in line with the actual situation of the smart city. In addition, the upload, download, and control protocol transport time can be considered to make the communication model better. In addition, collaboration between multiple MEC servers is also worth considering. For the Q-learning algorithm, if the environment changes, it needs to iterate many times to get the NE point, resulting in a low efficiency. In the future, combined with the neural network or any other learning methods, it may improve the performance of the algorithm.

Author Contributions: Software, T.L.; Supervision, F.X.; Validation, T.L.; Visualization, R.L.; Writing—original draft, T.L. and R.L.; Writing—review and editing, F.X., C.F. and C.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Research Innovation Fund for College Students of Beijing University of Posts and Telecommunications.

Acknowledgments: The authors are thankful to the Research Innovation Fund for College Students of Beijing University of Posts and Telecommunications for the support of research. In addition, the authors would like to thank the editor and reviewers for their valuable comments and suggestions that improved the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Founoun, A.; Hayar, A. Evaluation of the concept of the smart city through local regulation and the importance of local initiative. In Proceedings of the 2018 IEEE International Smart Cities Conference (ISC2), Kansas City, MO, USA, 16–19 September 2018. [CrossRef]
2. Mora, O.B.; Rivera, R.; Larios, V.M.; Beltran-Ramirez, J.R.; Maciel, R.; Ochoa, A. A Use Case in Cybersecurity based in Blockchain to deal with the security and privacy of citizens and Smart Cities Cyberinfrastructures. In Proceedings of the 2018 IEEE International Smart Cities Conference (ISC2), Kansas City, MO, USA, 16–19 September 2018. [CrossRef]
3. Martino, B.D.; Rak, M.; Ficco, M.; Esposito, A.; Maisto, S.; Nacchia, S. Internet of things reference architectures, security and interoperability: A survey. *Internet Things* **2018**, *1*, 99–112. [CrossRef]
4. Štefanič, P.; Cigale, M.; Jones, A.C.; Knight, L.; Taylor, I.; Istrate, C.; Suciu, G.; Ulisses, A.; Stankovski, V.; Taherizadeh, S.; et al. SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications. *Future Gener. Comput. Syst.* **2019**, *99*, 197–212. [CrossRef]
5. Kakderi, C.; Komninos, N.; Tsarchopoulos, P. Smart cities and cloud computing: lessons from the STORM CLOUDS experiment. *J. Smart Cities* **2016**, *1*, 4–13. [CrossRef]
6. Ciobanu, R.I.; Dobre, C.; Balanescu, M.; Suciu, G. Data and Task Offloading in Collaborative Mobile Fog-Based Networks. *IEEE Access* **2019**, *7*, 104405–104422. [CrossRef]
7. Patel, M. *Mobile-Edge Computing-Introductory Technical White Paper, ETSI MEC White Paper*; Technical Report; ETSI: Sophia-Antipolis, France, 2014; 36p.
8. Taleb, T.; Dutta, S.; Ksentini, A.; Iqbal, M.; Flinck, H. Mobile Edge Computing Potential in Making Cities Smarter. *IEEE Commun. Mag.* **2017**, *55*, 38–43. [CrossRef]
9. Pradhan, M.A.; Patankar, S.; Shinde, A.; Shivarkar, V.; Phadatare, P. IoT for smart city: Improvising smart environment. In Proceedings of the 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS 2017), Chennai, India, 1–2 August 2017. [CrossRef]
10. Zeng, Z.; Veeravalli, B.; Khan, S.U.; Teo, S.G. Cloud-of-clouds based resource provisioning strategy for continuous write applications. In Proceedings of the 2017 23rd Asia-Pacific Conference on Communications (APCC), Perth, Australia, 11–13 December 2017. [CrossRef]
11. Huang, C.M.; Chiang, M.S.; Dao, D.T.; Su, W.L.; Xu, S.; Zhou, H. V2V Data Offloading for Cellular Network Based on the Software Defined Network (SDN) Inside Mobile Edge Computing (MEC) Architecture. *IEEE Access* **2018**, *6*, 17741–17755. [CrossRef]
12. Berno, M.; Alcaraz, J.J.; Rossi, M. On the Allocation of Computing Tasks under QoS Constraints in Hierarchical MEC Architectures. In Proceedings of the 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC), Rome, Italy, 10–13 June 2019. [CrossRef]
13. Dinh, T.Q.; La, Q.D.; Quek, T.Q.S.; Shin, H. Learning for Computation Offloading in Mobile Edge Computing. *IEEE Trans. Commun.* **2018**, *66*, 6353–6367. [CrossRef]
14. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
15. Ranadheera, S.; Maghsudi, S.; Hossain, E. Mobile Edge Computation Offloading Using Game Theory and Reinforcement Learning. *arXiv* **2017**, arXiv:1711.09012.
16. He, Y.; Yu, F.R.; Zhao, N.; Leung, V.C.M.; Yin, H. Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep Reinforcement Learning Approach. *IEEE Trans. Commun.* **2017**, *55*, 31–37. [CrossRef]
17. Qian, L.P.; Wu, Y.; Ji, B.; Huang, L.; Tsang, D.H.K. HybridIoT: Integration of Hierarchical Multiple Access and Computation Offloading for IoT-Based Smart Cities. *IEEE Netw.* **2019**, *33*, 6–13. [CrossRef]

18. Ning, Z.; Xia, F.; Ullah, N.; Kong, X.; Hu, X. Vehicular Social Networks: Enabling Smart Mobility. *IEEE Commun. Mag.* **2017**, *55*, 16–55. [CrossRef]
19. Esposito, C.; Castiglione, A.; Palmieri, F.; Ficco, M.; Dobre, C.; Iordache, G.V.; Pop, F. Event-based sensor data exchange and fusion in the Internet of Things environments. *J. Parallel Distrib. Comput.* **2018**, *118*, 328–343. [CrossRef]
20. Sapienza, M.; Guardo, E.; Cavallo, M.; Torre, G.L.; Leombruno, G.; Tomarchio, O. Solving Critical Events through Mobile Edge Computing: An Approach for Smart Cities. In Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP), St. Louis, MO, USA, 18–20 May 2016. [CrossRef]
21. Zhao, L.; Wang, J.; Liu, J.; Kato, N. Optimal Edge Resource Allocation in IoT-Based Smart Cities. *IEEE Network* **2019**, *33*, 30–35. [CrossRef]
22. Deng, Y.; Chen, Z.; Yao, X.; Hassan, S.; Wu, J. Task Scheduling for Smart City Applications Based on Multi-Server Mobile Edge Computing. *IEEE Access* **2019**, *7*, 14410–14421. [CrossRef]
23. Zhou, Y.; Yeoh, P.L.; Pan, C.; Wang, K.; Elkashlan, M.; Wang, Z.; Vucetic, B.; Li, Y. Offloading Optimization for Low-Latency Secure Mobile Edge Computing Systems. *IEEE Wirel. Commun. Lett.* **2019**, 1. [CrossRef]
24. Fan, C.; Li, B.; Zhao, C.; Guo, W.; Liang, Y.C. Learning-Based Spectrum Sharing and Spatial Reuse in mm-Wave Ultradense Networks. *IEEE Trans. Veh. Technol.* **2018**, *67*, 4954–4968. [CrossRef]
25. Han, Z.; Niyato, D.; Saad, W.; Basar, T.; Hjorungnes, A. *Game Theory in Wireless and Communication Networks*; Cambridge University Press: Cambridge, UK, 2009.
26. Monderer, D.; Shapley, L.S. Potential Games. *Games Econ. Behav.* **1996**, *14*, 124–143. [CrossRef]
27. Claus, C.; Boutilier, C. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI , USA, 26–30 July 1998; pp. 746–752.
28. Apt, K.R.; Simon, S. A Classification of Weakly Acyclic Games. *arXiv* **2015**, arXiv:1206.0130.
29. Chapman, A.C.; Leslie, D.S.; Rogers, A.; Jennings, N.R. Convergent Learning Algorithms for Unknown Reward Games. *SIAM J. Control Optim.* **2013**, *51*, 3154–3180. [CrossRef]
30. Perez-Romero, J.; Sanchez-Gonzalez, J.; Agusti, R.; Lorenzo, B.; Glisic, S. Power-Efficient Resource Allocation in a Heterogeneous Network With Cellular and D2D Capabilities. *IEEE Trans. Veh. Technol.* **2016**, *65*, 9272–9286. [CrossRef]