Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8836592, 11 pages
https://doi.org/10.1155/2020/8836592

WILEY | Hindawi

Research Article

A Smart Cache Content Update Policy Based on Deep Reinforcement Learning

Lincan Li,[1] Chiew Foong Kwong,[1] Qianyu Liu,[2] and Jing Wang[1]

[1]*Department of Electrical and Electronic Engineering, University of Nottingham Ningbo China, 315100 Ningbo, China*
[2]*International Doctoral Innovation Centre, University of Nottingham Ningbo China, 315100 Ningbo, China*

Correspondence should be addressed to Chiew Foong Kwong; chiew-foong.kwong@nottingham.edu.cn

Received 10 May 2020; Revised 10 October 2020; Accepted 15 October 2020; Published 9 November 2020

Academic Editor: Lisheng Fan

Copyright © 2020 Lincan Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper proposes a DRL-based cache content update policy in the cache-enabled network to improve the cache hit ratio and reduce the average latency. In contrast to the existing policies, a more practical cache scenario is considered in this work, in which the content requests vary by both time and location. Considering the constraint of the limited cache capacity, the dynamic content update problem is modeled as a Markov decision process (MDP). Besides that, the deep Q-learning network (DQN) algorithm is utilised to solve the MDP problem. Specifically, the neural network is optimised to approximate the Q value where the training data are chosen from the experience replay memory. The DQN agent derives the optimal policy for the cache decision. Compared with the existing policies, the simulation results show that our proposed policy is 56%–64% improved in terms of the cache hit ratio and 56%–59% decreased in terms of the average latency.

1. Introduction

The recent rapid evolution of mobile communication techniques and the proliferation of smart mobile devices have caused an exponential growth in mobile network traffic [1] [2]. According to Cisco [3], global mobile network traffic will reach 77 exabytes each month by 2022. As such, it will lead to data traffic congestion of the backhaul [4]. To mitigate this, a cache-enabled technique has emerged that is regarded as an effective method that can alleviate data traffic congestion [5]. In a cache-enabled network, a portion of the popular content is cached at the edge of the network at base stations (BSs) or user terminals (UTs), where users can directly access and download the cached content from the edge rather than from the core network via backhaul links. Consequently, data traffic congestion of the backhaul can be reduced and content retrieval from the edge can be faster than from the remote core network [6, 7].

However, because of the limited cache capacity, it is necessary to update cache content to ensure that cache-enabled networks always store the most popular content [8]. The most two common content update policies are the least frequently used (LFU) policy and the least recently used (LRU) policy [9]. LRU frequently stores the content with the latest access time, and LFU frequently stores content with the largest cumulative request times. Besides, as described in [10], a heterogeneous cache structure is proposed, in which the most popular contents are stored at small BSs and the less popular contents are stored at macro BSs. The combination of small BSs and macro BSs can maximise the network capacity and satisfy the content transmission demand. In [11], an optimal cooperative cache policy that can increase the cache hit ratio was presented. The cache hit ratio is utilised to describe how frequently content is requested by mobile users. In [9], an adaptive cache policy was proposed that can reduce user access latencies. In [12], an edge cache policy was proposed to reduce the average content delivery latency. However, conventional methods lack adaptive ability in dynamic cache scenarios. The reason is that they assume that the content popularity distribution is known or can be accurately predicted, which is difficult to achieve in dynamic caching scenarios. In this case, due to an inaccurate distribution of

content popularity, the conventional methods have poor cache performances, since their performances are highly dependent on the accurate distribution of content popularity.

Motivated by the deep reinforcement learning (DRL) approach in solving the dynamic problem [13], DRL has been applied into cache policies to improve the cache performance of dynamic cache scenarios. In [14], a DRL approach was proposed to reduce the transmission cost by jointly considering proactive cache and content recommendations. In [15], a cache content update policy based on DRL was proposed to improve energy efficiency. In [16], a DRL model was utilised to minimise transmission latencies. For specific, reinforcement learning (RL) is applied to obtain the optimal cache policy. In [17], a DRL-based policy was proposed to minimise system power consumption. In [18], a deep Q-learning network (DQN) algorithm, one branch of DRL, is applied to do the network slicing decision and allocates the spectrum resources for the content delivery. In [19], a DQN-based mobile edge computing network is proposed, in which several computation tasks are offloaded from the user terminals to the computational access points. Although DQN has attracted significant attention in the cache-enabled network, there are very little works done in applying DQN into the cache content update phase. Moreover, most of the previously mentioned DRL-based cache policies assume the content requests as a time-varying variable. They did not adopt more practical scenarios in which the content requests are varied in both time and location, also known as spatiotemporally varying scenarios.

Inspired by the aforementioned literature, in this paper, a DQN-based content update policy at BSs is proposed to increase the cache hit ratio and reduce average latency, as well as considering spatiotemporally varying scenarios in which content requests vary by both time and location. The reasons to apply DQN are as follows: (1) DQN has a faster convergence speed than the conventional DRL policies, e.g., advanced actor-critic (A2C) and deep deterministic policy gradient (DDPG) [14]. (2) DQN can adapt to the varying scenarios, as long as the dynamic problem is correctly modeled and the DQN agent is allowed to continuously learn experience from the environment [18]. The main contributions are summarised as follows:

(i) The dynamic cache content update problem is formulated as a Markov decision process (MDP) problem, which is solved by a DQN algorithm. Specifically, the neural network is utilised to approximate the Q value and the DQN agent is used to decide whether or not to cache the requested content

(ii) Our proposed policy is compared with LRU, LFU and DRL [20] policies and the simulation results demonstrate that our proposed policy has the best cache performance in terms of the cache hit ratios and average latencies

The rest of this paper is organised as follows. The system model and problem formulation are introduced in Section 2. The detailed elements of the MDP framework and the principles of the DQN-based cache content update policy are discussed in Section 3. The simulation results are shown in Section 4, and the conclusion is provided in Section 5.

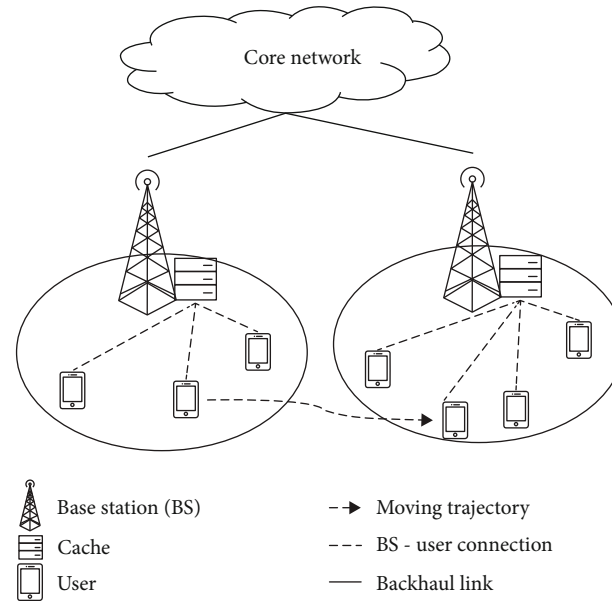

Figure 1: The system model of the cache-enabled network.

2. System Model and Problem Formulation

In this section, the system model and the problem of how to maximise the cache hit ratio and minimise average latency are introduced.

2.1. System Model. As shown in Figure 1, the cache-enabled system includes one core network, $\mathscr{M}$ cache-enabled BSs, and $\mho$ mobile users. Each BS can store $\mathscr{H}$ contents at most. The total content library $\mathscr{W} = \{1, 2, \cdots, \omega\}$ contains ω kinds of contents and each content has the same size Size_{df}. The core network is assumed that has enough capacity to store the entire contents. Each BS covers a circular cellular region with a fixed radius, and all of the mobile users in its cellular region can connect with the serving BS (the BS where users connect). Mobile users can directly retrieve their requested content from the serving BS if the content is cached locally (the requested content is already cached at the serving BS); otherwise, the requested content must be retrieved from the core network. The i^{th} BS is regarded as a DQN agent and receives the spatiotemporal content requests $R^i = \{R_1^i, R_2^i, \cdots, R_t^i, \cdots\}$, where R_t^i is the current content request at the i^{th} BS. From the received content requests, the DQN agent can decide when and where (which BS) to cache the content or not. If cached, the DQN agent further decides which cached content is replaced by the currently requested content; otherwise, the cached contents remain the same. The action space of the i^{th} BS is defined as $A^i = \{A_0^i, A_1^i, A_2^i, \cdots, A_{\mathscr{H}}^i\}$ and A^i uses one hot code. $A_0^i = 1$ means that the cached content remains the same, and $A_v^i = 1$ means that the v^{th} cached content is replaced by the currently requested content, where $v \in \{1, 2, \cdots, \mathscr{H}\}$. In summary, at each time slot

t, each BS receives numerous content requests including the user preference content and location information, and each DQN agent executes one action from the corresponding action space to maximise the cache hit ratio and minimise the average latency.

2.2. Problem Formulation. The problem in this study consists of two subproblems: maximising the cache hit ratio and minimising average latency.

2.2.1. Maximising the cache hit ratio. The cache hit ratio is utilised to describe the probability of the requested content at the local cache. The system cache hit ratio $\mathscr{P}_{\text{hit\_ratio}}$ is formulated for N requests as follows:

$$\mathscr{P}_{\text{hit\_ratio}} = \frac{\sum_{i=1}^{\mho}\sum_{t=1}^{N} F\left(R_t^i\right)}{\mho \times N}, \tag{1}$$

where $F(R_t^i)$ is a function to test whether the requested content is cached locally. The definition of $F(R_t^i)$ is as follows:

$$F\left(R_t^i\right) = \begin{cases} 1, \text{if } R_t^i \text{ is cached locally} \\ 0, \text{otherwise} \end{cases} \tag{2}$$

Maximising the cache hit ratio is expressed as follows:

$$\begin{aligned} &\boldsymbol{P\_1}: \text{Max } \mathscr{P}_{\text{hit\_ratio}} \\ &s.t. \sum_{t=1}^{T} F\left(R_t^i\right) \le \mathscr{H} \end{aligned} \tag{3}$$

2.2.2. Minimising the average latency. The latency is an indicator that evaluates the cache content update policy's performance. The latency is the time when content is transmitted from one location to another. The latency consists of the transmission latency $\mathscr{T}_{\text{tr}}$, propagation latency $\mathscr{T}_{\text{prp}}$, processing latency $\mathscr{T}_{\text{pro}}$, and queue latency $\mathscr{T}_{\text{qu}}$. From [20], the expression of the latency $\mathscr{T}$ is given as:

$$\mathscr{T} = \mathscr{T}_{\text{tr}} + \mathscr{T}_{\text{prp}} + \mathscr{T}_{\text{pro}} + \mathscr{T}_{\text{qu}}. \tag{4}$$

Normally in the content update process, the destination of the content packet is determinate, and the content packet is assumed that does not need to wait for transmission. Hence, the processing and queue latencies can be neglected during the content update process [20, 21], and the expression of the latency can be optimised as follows:

$$\begin{aligned} \mathscr{T}_{\text{tr}} &= \frac{\text{Size}_{\text{df}}}{v_{\text{tr}}}, \\ \mathscr{T}_{\text{prp}} &= D^* \times \frac{d}{\mathscr{R}}, \\ \mathscr{T} &= \mathscr{T}_{\text{tr}} + \mathscr{T}_{\text{prp}} = \frac{\text{Size}_{\text{df}}}{v_{\text{tr}}} + D^* \times \frac{d}{\mathscr{R}}, \end{aligned} \tag{5}$$

where Size_{df} is the content size, v_{tr} is the content transmission rate, $\mathscr{R}$ is the maximal coverage radius of the serving BS or core network, d is the distance between the user and the serving BS or between the serving BS and the core network, and D^* is the maximal propagation latency between the user and the serving BS or between the serving BS and the core network. To meet the requirement of the fifth-generation (5G) communication [22], the indicator D^* is expressed as follows:

$$D^* = \begin{cases} D^*_{\text{user-BS}} = 0.5 \sim 1.5 \text{ ms, if it is a user} - \text{BS connection} \\ D^*_{\text{BS-core}} = 10 \sim 20 \text{ ms, if it is a BS} - \text{core network connection} \end{cases}, \tag{6}$$

where $D^*_{\text{user-BS}}$ is the maximal propagation latency between the user and the serving BS, and $D^*_{\text{BS-core}}$ is the maximal propagation latency between the serving BS and the core network.

In more detail, if the requested content is cached locally, the content can be directly retrieved from the serving BS. Thus, for a hit content request, we consider the maximal propagation latency between the user and the serving BS $D^*_{\text{user-BS}}$, the distance between the user and the serving BS $d_{\text{user-BS}}$, and the maximal coverage radius of the serving BS $\mathscr{R}_{\text{BS}}$. The definition of the hit content $\mathscr{T}_{\text{hit}}$ latency is as follows:

$$\mathscr{T}_{\text{hit}} = \frac{\text{Size}_{\text{df}}}{v_{\text{tran}}} + D^*_{\text{user-BS}} \times \frac{d_{\text{user-BS}}}{\mathscr{R}_{\text{BS}}}. \tag{7}$$

If the requested content is missed at the serving BS, the serving BS needs to first retrieve the requested content from the core network and then deliver the requested content to the corresponding user. Hence, for a missed content request, we consider the maximal propagation latency between the user and the serving BS $D^*_{\text{user-BS}}$, the maximal propagation latency between the serving BS and the core network $D^*_{\text{BS-core}}$, the distance between the user and the serving BS $d_{\text{user-BS}}$, the distance between the serving BS and the core network $d_{\text{BS-core}}$, the maximal coverage radius of the serving BS $\mathscr{R}_{\text{BS}}$, and the maximal coverage radius of the core network $\mathscr{R}_{\text{core}}$. The definition of the latency of missed content $\mathscr{T}_{\text{miss}}$ is as follows:

$$\mathscr{T}_{\text{miss}} = \frac{\text{Size}_{\text{df}}}{v_{\text{tran}}} + D^*_{\text{user-BS}} \times \frac{d_{\text{user-BS}}}{\mathscr{R}_{\text{BS}}} + D^*_{\text{BS-core}} \times \frac{d_{\text{BS-core}}}{\mathscr{R}_{\text{core}}}. \tag{8}$$

The system latency $\mathscr{T}_{\text{sy}}$ is the sum of the latency of all of the hit content requests and all of the missed content requests. The average latency $\mathscr{T}_{\text{ave}}$ is the system latency divided by the number of content requests E. The $\mathscr{T}_{\text{sy}}$ and $\mathscr{T}_{\text{ave}}$ are defined as follows:

$$\begin{aligned} \mathscr{T}_{\text{sy}} &= \left[E \times \mathscr{P}_{\text{hit}_{\text{ratio}}} \times \mathscr{T}_{\text{hit}} + E \times \left(1 - \mathscr{P}_{\text{hit}_{\text{ratio}}}\right) \times \mathscr{T}_{\text{miss}}\right], \\ \mathscr{T}_{\text{ave}} &= \left[\mathscr{P}_{\text{hit}_{\text{ratio}}} \times \mathscr{T}_{\text{hit}} + \left(1 - \mathscr{P}_{\text{hit}_{\text{ratio}}}\right) \times \mathscr{T}_{\text{miss}}\right]. \end{aligned} \tag{9}$$

The problem on how to minimise the average latency can be formulated as follows:

$$\begin{aligned} \boldsymbol{P_2}: \text{Min } \mathscr{T}_{\text{ave}}, \\ s.t. \mathscr{P}_{\text{hit\_ratio}} \in [0, 1]. \end{aligned} \tag{10}$$

3. A Deep Q-Learning Network-Based Cache Content Update Policy

The related elements of the deep Q-learning network will be introduced in Section 3.1. The principle of the DQN algorithm and the workflow of our proposed cache policy will be provided in Section 3.2.

3.1. The Description of the Related Elements of the Deep Q-Learning Network. The principle of the DQN can be regarded as a Markov decision process (MDP) [23, 24]. To apply the DQN to the cache content update problem, the related notations under the DQN framework are described.

3.1.1. State Space. In time slot t, the instant state consists of the currently cached content, the currently requested content and its corresponding user, the user's next location, and the current time. In time slot t, the current instant state s_t is defined as

$$s_t = \left\{c_t^i, R_t^i, j_t, L_t^j\right\}, \tag{11}$$

where c_t^i is the cached content at the i^{th} DQN agent, R_t^i is the currently requested content, j_t is the unique name of the mobile user currently requesting the content, L_t^j is the next location of the j^{th} user, $i \in \{1, 2, \cdots, \mathscr{M}\}$, and $j \in \{1, 2, \cdots, \mho\}$.

The state space S is the set of all of the instant states over a time period. It is defined as

$$S = \{s_0, s_1, s_2, \cdots, s_t, \cdots\}. \tag{12}$$

3.1.2. Action Space. In each time slot t, the i^{th} DQN agent decides whether or not to cache the currently requested content. If yes, the DQN agent decides which cached content is replaced by the currently requested content; otherwise, the cached content remains the same. The action space of the i^{th} DQN A^i is defined as

$$A^i = \left\{A_0^i, A_1^i, A_2^i, \cdots, A_{\mathscr{H}}^i\right\}, \tag{13}$$

where A^i uses one hot code, which means only one action can be executed in a time slot. In this study, $A_0^i = 1$ means the cached content remains the same and $A_v^i = 1$ means the v^{th} cached content is replaced by the currently requested content, where $v \in \{1, 2, \cdots, \mathscr{H}\}$ and $\mathscr{H}$ is the maximal capacity of the i^{th} BS.

3.1.3. Reward and Value Functions. The reward r_t is the instant cache hit ratio in the time slot t. Specifically, reward $r_t = 1$ when the currently requested content is hit in the next state s_{t+1}; otherwise, $r_t = 0$. The policy $\pi(s) = \mathscr{P}(a \mid s_t)$ is a map that shows the probability of the execution of action a_t under the current state s_t, and $a_t \in A^i$. The MDP evaluates and optimises the policy based on the value function, which is defined as the expected value of cumulative discounted rewards received over the entire process following the policy [25]. There are two definitions of value functions: one is the state value function and the other is the state-action value function. The state value function is the expected value of a discounted cumulative reward in the current state s_t when the agent follows the policy. The state value function is defined as follows:

$$V_\pi(s) = E_\pi\left[\sum_{u=0}^{\infty} \gamma^u r_{t+u+1} \mid s_t\right]. \tag{14}$$

The state-action value function is the expected value of the discounted cumulative reward from the current state s_t and action a_t is based on the policy used to choose one action. The definition of the state-action value function is

$$Q_\pi(s, a) = E_\pi\left[\sum_{u=0}^{\infty} \gamma^u r_{t+u+1} \mid (s_t, a_t)\right], \tag{15}$$

where $\gamma \in [0, 1]$ is a discount factor that affects the future reward from the current state s_t. The target of the MDP is finding the optimal $\pi(s)$ and $\pi^*(s)$ that can obtain the maximal value function.

3.2. The Cache Content Update Based on the Deep Q-Learning Network

3.2.1. Principle of the DQN Framework. DQN is an effective hybrid framework of neural networks and Q-learning. In this framework, the neural network is applied to predict the Q values rather than recording the Q values in a Q table. However, the DQN will not be efficient when considering only the combination of Q-learning and the neural network. The following two characteristics improve the DQN framework's efficiency.

(i) The DQN has two neural networks with the same structures operating in different parameters, the evaluation network and the target network. The parameters of the evaluation and target networks are defined as θ and θ^-, respectively. The evaluation network uses the latest parameter θ to predict the current state-action Q values $Q_{(s_t, a_t, \theta)}$, where θ is updated in each iteration. The target network uses the parameter θ^- to predict the next state-action Q value $Q_{(s_{t+1}, a_{t+1}, \theta^-)}$, where θ^- is updated over a period time. The target network can solve the correlation of the Q value with the Q target value, which makes the DQN easier to converge

(ii) DQN has an experience replay memory with a limited capacity. The current state s_t, action a_t, reward r_t, and next state s_{t+1} are stored in format (s_t, a_t,

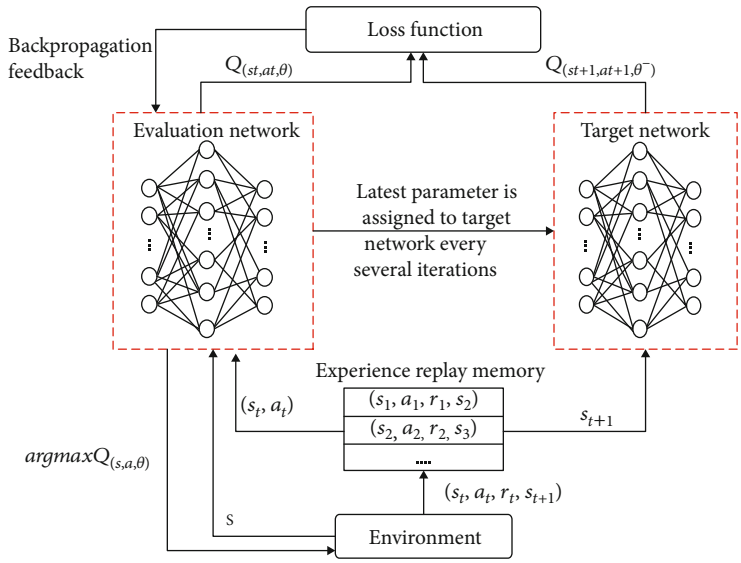

Figure 2: Flow chart of the deep Q-learning network.

r_t, and s_{t+1}) into the memory as experiences. Once the capacity is full, new received experiences will replace earlier experiences. During the training stage, the training data are randomly selected from the experience replay memory. The random selection disorganises the experience correlation, which solves the neural network's overfitting issue

The neural network enables $Q_{(s_t,a_t,\theta)} \approx Q_{(s_t,a_t)}$ [26]. According to [5], the evaluation of $Q_{(s_t,a_t)}$ is derived from Q-learning as:

$$Q_{(s_t,a_t)} = Q_{(s_t,a_t)} + \alpha\left[r_t + \gamma * \max Q_{(s_{t+1},a_{t+1})} - Q_{(s_t,a_t)}\right], \tag{16}$$

where α is the learning rate $\epsilon\,(0, 1)$, and γ is the discount factor $\epsilon\,[0, 1]$.

The neural network can be trained via the minimisation of the loss function. The loss function Loss (θ) is defined as:

$$\begin{aligned}\text{Loss}\,(\theta) &= E\left[\left(Q_{(s_{t+1},a_{t+1},\theta^-)} - Q_{(s_t,a_t,\theta)}\right)^2\right] \\ &= E\left[\left(r_t + \gamma * \max Q_{(s_{t+1},a_{t+1},\theta^-)} - Q_{(s_t,a_t,\theta)}\right)^2\right],\end{aligned} \tag{17}$$

where $r_t + \gamma * \max Q_{(s_{t+1},a_{t+1},\theta^-)}$ is the target network's Q value and $Q_{(s_t,a_t,\theta)}$ is the evaluation's Q value.

The detailed optimisation of the evaluation network and target network is shown in Figure 2. In each training step, the evaluation network receives a backpropagated loss function based on a batch of experiences randomly selected from the experience replay memory. The parameter of the evaluation network θ is then updated by the minimisation of the loss function via the stochastic gradient descent (SGD) function. After several steps, the parameter of the target network θ^- is updated by assigning the latest parameter θ to θ^-. After a training period, the two neural works are stably trained.

3.2.2. The Workflow of the Cache Content Update Policy Based on DQN. In each decision epoch, the i^{th} DQN agent receives a content request. If the content is cached locally, the serving BS delivers the requested content to the corresponding user. If the content is missed at the serving BS, the serving BS retrieves the requested content from the core network and then delivers the content to the corresponding user. Subsequently, the requested content is cached at the serving BS when the cache capacity is not full. If the cache capacity is full, the optimised evaluation network outputs the Q value of all of the actions, and the DQN agent selects an action a_k with the maximal Q value. After the execution of the action a_k, the new instant reward is calculated into the target network's Q value and a new loss function is obtained based on Eq. (17). The parameters θ and θ^- are then updated based on the minimisation of the new loss function. After a training period, the best policy $\pi^*(s)$ that can maximise the cache hit ratio and minimise the average latency is derived. The DQN-based cache content update policy is shown in Algorithm 1.

4. Results and Discussion

In this study, we consider a cache-enabled network with 4 BSs and 10 mobile users and ensure that each user is covered by a BS. For simplicity, the users are distributed along with the edge of the serving BS, and each BS has the maximal communication distance with the core network, and hence, the rate $\mathcal{A}/\mathcal{R}$ is 1. Besides, there is no overlap between any two BSs to avoid the handover between any two BSs. Furthermore, each content has the same size (2,000 bits), and the content transmission rate is 35 Mbit/s. The neural network has three layers, the input layer, hidden layer, and output layer. The hidden layer has 512 neurons, and the number of

The DQN-based cache content update algorithm.
Input: The feature of the state s_t
Initialise the parameter θ and θ^- and instant reward r_t =0
for step =1, Y ***do***
  for t = 1, T ***do***
    Receive a content request
    if the content request is cached locally, ***then***
    BS directly delivers the requested content to the user ***end epoch***
    elif
    The cache capacity is not full, ***then***
    BS retrieves the requested content from the core network and delivers the requested content to the user
    The requested content is cached locally ***end epoch***
    elif
    The cache capacity is full, ***then***
    observe the current state s_t
    randomly generate a value ρ
    if $\rho < \varepsilon$, ***then***
      randomly select an action a_t from the action spaces
      else
      $a_t = \text{argmax}Q_{(s_t, a_t, \theta)}$
      end if
      execute a_t, receive the reward r_t, next state s_{t+1}
      store (s_t, a_t, r_t, s_{t+1}) into the experience replay memory
      randomly selects a mini-batch of the experiences
      update the parameter of the evaluation θ via the minimisation of the backpropagated loss
      update the parameter of the evaluation θ^- in several time slots
      end if
  end for
end for

Algorithm 1:

neurons at the input and output layers is $(\mathcal{H} + 3)$ and $(\mathcal{H} + 1)$, respectively. The maximal cache capacity $\mathcal{H}$ is described in each experiment. The learning rate ℓ is 0.9, the greedy parameter ε is 0.9, and the discount factor γ is 0.1. The content requests of the i^{th} user are generated following the Zipf distribution law as

$$p(\delta, \mathcal{K}, B) = \frac{\delta^{-\mathcal{K}}}{\sum_{b=1}^{B} b^{-\mathcal{K}}}, \tag{18}$$

where δ is the content rank, $\mathcal{K}$ is the Zipf parameter, and B is the total number of content requests. In each experiment, we assume that the total number of content requests is 7,200.

Figure 3 investigates the cache hit ratios of the LFU policy, LRU policy, DRL policy in [20], and our proposed policy. The Zipf parameters vary from 1.1 to 1.8, the users' locations are fixed, and the cache can store 288 types of contents at most. As the Zipf parameter $\mathcal{K}$ increases, the four policies' cache hit ratios increase. This occurs because as the Zipf parameter increases, there is less content with larger probabilities of content requests. In other words, the popular content becomes more popular, the unpopular content becomes less popular, and the type of content decreases. Considering the same cache capacity, the cached content is more popular, and therefore, the cache hit ratio increases. Our proposed policy has the highest cache hit ratio regardless of the Zipf parameter. The simulation demonstrates that the effect of the popular content in the cache hit ratio increases as the Zipf parameter increases. Thus, our proposed policy is superior to the three other policies.

Figure 4 investigates the effect of the cache capacity on the cache hit ratio. Here, the Zipf parameter is 1.4, and the mobile users' locations are fixed. The varied cache capacity is 36, 72, 108, 144, 180, 216, 252, and 288. As demonstrated, our proposed policy is superior to the three other policies since our proposed policy has the highest cache hit ratio. In addition, as the cache capacity increases, the cache hit ratios of the four policies continuously increase. When the capacity is 288, the cache hit ratios of the four policies are remarkably close. This occurs because the popular content dominates the cache hit ratio, and the cache capacity is high enough to store all of the popular contents.

The cache hit ratio under spatiotemporally varying scenarios is shown in Figure 5. In the experiment, the cache can store 216 types of contents at most, the Zipf parameters are randomly generated from 1.2 to 1.6 every 20,000 time slots, and the users are initially fixed and randomly change their locations among the four BSs after the 20,000 time slot. When the users' locations and the Zipf parameters are fixed, the gaps between our proposed policy and the three other policies are gradually stable. This occurs because all of the policies are optimally trained. After time slot 20,000, the four policies immediately decrease. This occurs because the

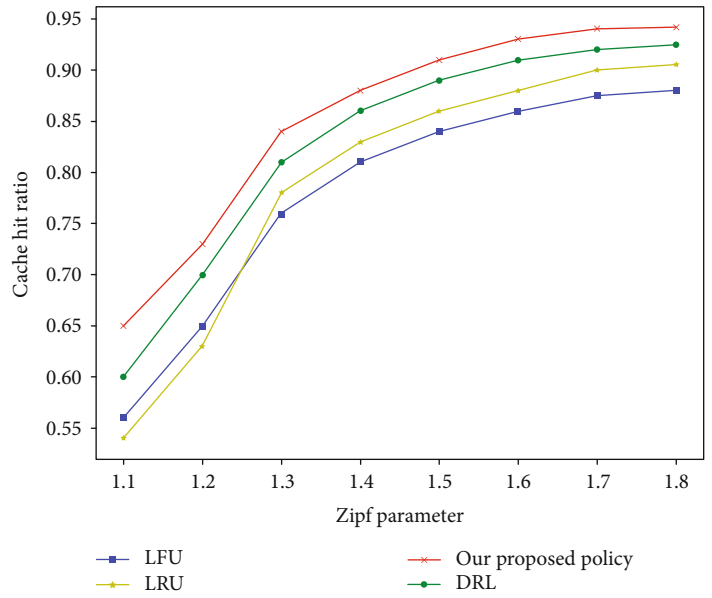

Figure 3: The cache hit ratio vs. the varying Zipf parameters. We assume that the Zipf parameters = 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, and 1.8.

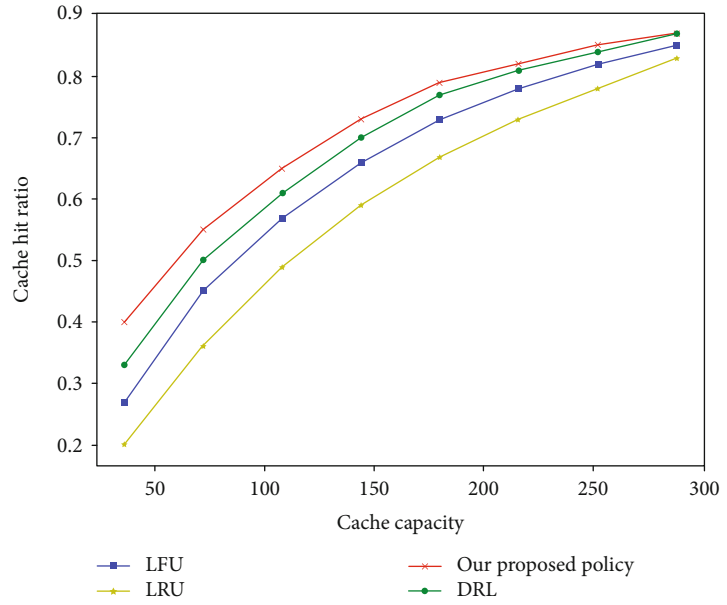

Figure 4: The cache hit ratio vs. the cache capacity. The varied cache capacity is 36, 72, 108, 144, 180, 216, 252, and 288.

content popularity changes with the random movement of the users and random generation of the Zipf parameters. Later, our proposed policy's curve slowly increases, while the three other policies' curves continuously decrease. The gaps between our proposed policy's curve and the other policies' curves continuously increase. Our proposed policy eventually improves by at least 56% compared with the three other policies. The growth ratio g is derived based on $g = C_{our} - C_{existing}/C_{existing}$, in which C_{our} and $C_{existing}$ is the cache hit ratio of our proposed policy and any one of the other three policies, respectively. This significant improvement occurs because our proposed policy considers the effect of the users' random distribution and the random generation of Zipf parameters. Therefore, our proposed policy quickly adapts to spatiotemporally varying content requests. Consequently, we conclude that our proposed policy is superior for managing spatiotemporally varying problems.

Figure 6 demonstrates the four policies' average latencies under different Zipf parameters. Here, the Zipf parameters vary from 1.1 to 1.8, the mobile users' locations are fixed, and the cache can store 288 types of contents at most. As demonstrated, our proposed policy always has the lowest cache hit ratio compared with the other three policies. Thus, our proposed policy has the best cache hit ratio. The higher

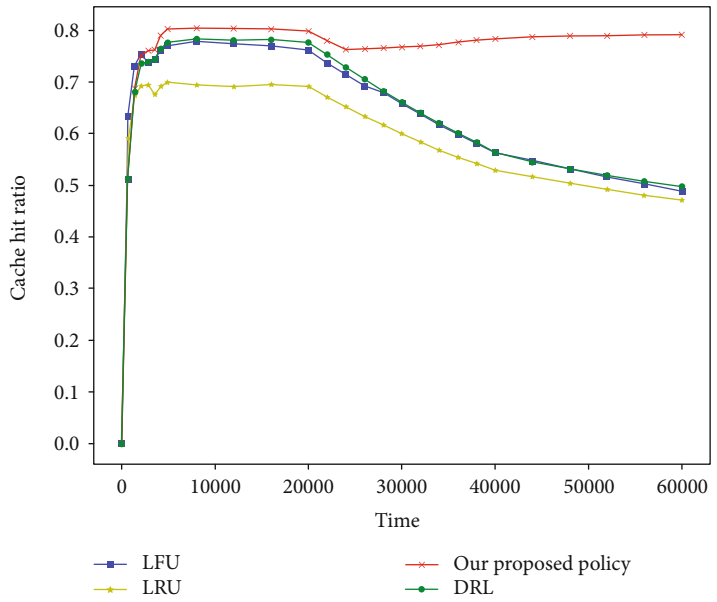

Figure 5: The cache hit ratio under spatiotemporally varying scenarios in which the mobile users randomly change their locations after the 20,000 time slot.

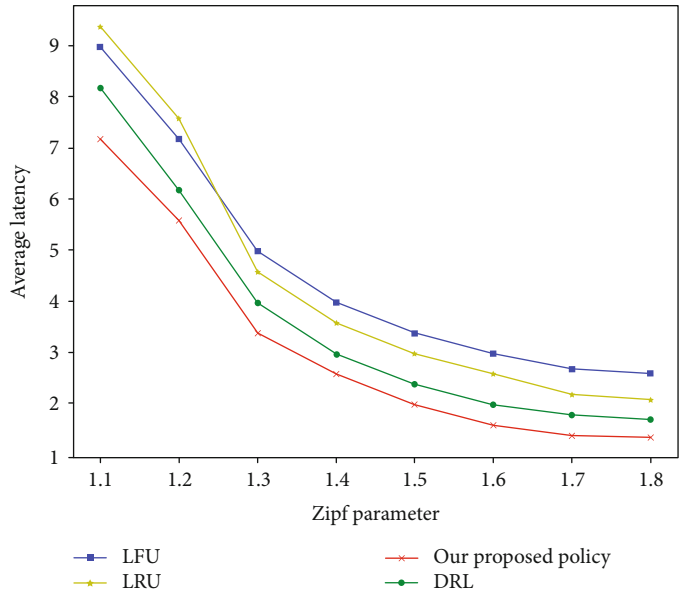

Figure 6: The average latency vs. the varying Zipf parameters. We assume that the Zipf parameters = 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, and 1.8.

the cache hit ratio is, the more contents can be retrieved locally. The local latency from the BS is much smaller than the remote latency from the core network. Therefore, our proposed policy performs better than the other three policies in terms of the average latency.

As shown in Figure 7, we investigate the effect of the cache capacity on the average latency. In this simulation, the Zipf parameter is 1.4, and the mobile users' locations are fixed. The cache capacity is 36, 72, 108, 144, 180, 216, 252, and 288. The higher the cache capacity, the lower the average latency of each policy. This occurs because more contents can be cached locally as the cache capacity increases. In addition, the slope of each policy gradually decreases. This occurs because all of the policies aim to cache the most popular contents via their limited cache capacity. As the cache capacity further increases, more contents are cached, while the recently cached contents are less popular than the initially cached contents. Consequently, the average latency increases less when caching less popular contents. Furthermore, our proposed policy has the minimal average latency regardless of the cache capacity.

Figure 8 shows the average latency under spatiotemporally varying scenarios. In the experiment, the cache is assumed that can store 216 types of contents at most, the Zipf

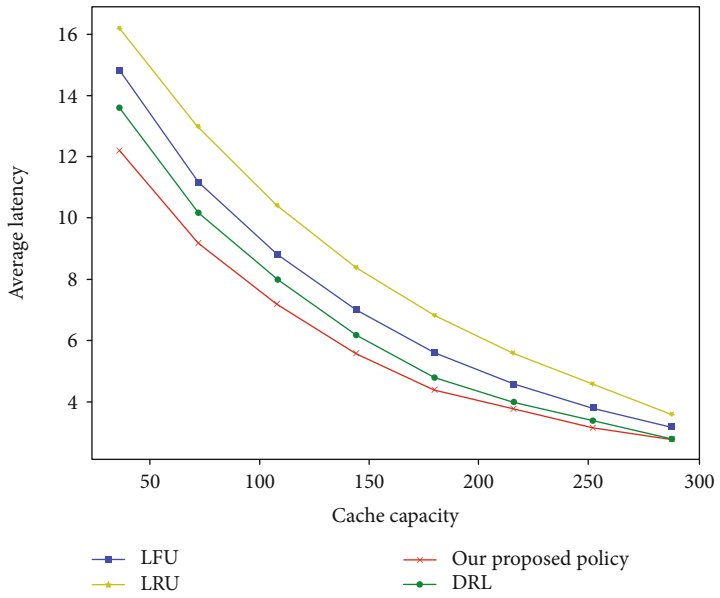

Figure 7: The average latency vs. the varying cache capacity. We assume that the cache capacity is 36, 72, 108, 144, 180, 216, 252, and 288.

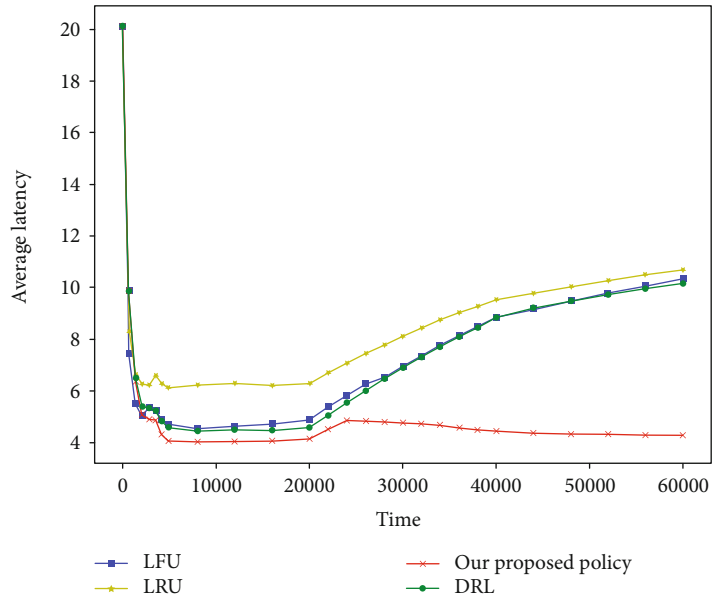

Figure 8: The average latency under spatiotemporally varying scenarios in which the mobile users randomly change their locations after the 20,000 time slot.

parameter is randomly generated from 1.2 to 1.6 every 20,000 time slots, and the users are initially fixed and randomly change their locations among the four BSs after the 20,000 time slot. In the first 20,000 time slots, each policy finally has a stable cache performance after a training period. Once the users randomly move among the four BSs, the LRU, LFU, and DRL policies' curves immediately increase, and our proposed policy's curve first slightly increases and then gradually decreases. More specifically, our proposed policy achieves a 56%-59% decrease compared to the three other policies. The reduction rate $\mathfrak{y}$ is derived based on $\mathfrak{y} = L_{\text{existing}} - L_{\text{our}}/L_{\text{existing}}$, in which L_{our} and L_{existing} is the latency of our proposed policy and any one of the other three policies, respectively. The decrease occurs because our proposed policy considers the effect of the dynamic changes in the user distribution and Zipf parameters on the latency, while the other three policies do not. The simulation demonstrates that our proposed policy can perform stably under spatiotemporally varying scenarios.

5. Conclusions

In this study, a DRL-based cache content update policy is proposed with the objective to maximise the cache hit ratio

and minimise the average latency. Compared to the existing policies, a more practical cache scenario is considered, in which the content requests vary spatiotemporally. The dynamic content update problem is formulated as an MDP problem, and DQN is applied to solve this MDP problem. Specifically, the neural network is trained to approximate the Q value, in which the training data are chosen from the experience replay memory. The DQN agent derives the optimal policy from the neural network for the cache decision. Compared with the existing policies, e.g., the LFU, LRU, and DRL [20] policies, the simulation results show that our proposed DRL-based cache content update policy has the best cache performance in the considered spatiotemporally varying scenario and is 56%–64% improved in terms of the cache hit ratio and 56%–59% decreased in terms of the average latency.

Data Availability

Content requests were described in the simulation section.

Conflicts of Interest

Lincan Li, Chiew Foong Kwong, Qianyu Liu, and Jing Wang declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was supported by Ningbo Natural Science Programme (NBNSP), project code 2018A610095.

References

[1] M. Chen, Y. Hao, L. Hu, K. Huang, and V. K. N. Lau, "Green and mobility-aware caching in 5G networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 12, pp. 8347–8361, 2017.

[2] D. Deng, J. Xia, L. Fan, and X. Li, "Link selection in buffer-aided cooperative networks for green IoT," *IEEE Access*, vol. 8, pp. 30763–30771, 2020.

[3] Cisco, *Cisco visual networking index: global mobile data traffic forecast update, 2017–2022*, White Paper, 2019.

[4] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, Princeton, NJ, USA, March 2018.

[5] Z. Zhao, W. Zhou, D. Deng, J. Xia, and L. Fan, "Intelligent mobile edge computing with pricing in Internet of Things," *IEEE Access*, vol. 8, pp. 37727–37735, 2020.

[6] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for Internet of Things: a deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2074–2083, 2019.

[7] Y. Wu, S. Yao, Y. Yang, Z. Hu, and C.-X. Wang, "Semigradient-based cooperative caching algorithm for mobile social networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Washington, DC, USA, December 2016.

[8] P. Wu, J. Li, L. Shi, M. Ding, K. Cai, and F. Yang, "Dynamic content update for wireless edge caching via deep reinforcement learning," *IEEE Communications Letters*, vol. 23, no. 10, pp. 1773–1777, 2019.

[9] X. Zhang and Y. Cao, "A cooperation-driven ICN-based caching scheme for mobile content chunk delivery at RAN," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1437–1442, Valencia, Spain, June 2017.

[10] S. Zhang, N. Zhang, P. Yang, and X. Shen, "Cost-effective cache deployment in mobile heterogeneous networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 11264–11276, 2017.

[11] R. Wang, F. Hajiaghajani, and S. Biswas, "Distributed caching in mobile networks with heterogeneous content demand," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 172–178, Las Vegas, NV, USA, Jan 2017.

[12] Z. Luo, M. LiWang, Z. Lin, L. Huang, X. Du, and M. Guizani, "Energy-efficient caching for mobile edge computing in 5G networks," *Applied Sciences*, vol. 7, no. 6, p. 557, 2017.

[13] D. Guo, L. Tang, X. Zhang, and Y.-C. Liang, "Joint optimization of handover control and power allocation based on multi-agent deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2020.

[14] D. Liu and C. Yang, "A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users," *IEEE Access*, vol. 7, pp. 83120–83136, 2019.

[15] W. Li, J. Wang, G. Zhang, L. Li, Z. Dang, and S. Li, "A reinforcement learning based smart cache strategy for cache-aided ultra-dense network," *IEEE Access*, vol. 7, pp. 39390–39401, 2019.

[16] Y. Wei, Z. Zhang, F. R. Yu, and Z. Han, "Joint user scheduling and content caching strategy for mobile edge networks using deep reinforcement learning," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, Kansas City, MO, USA, May 2018.

[17] Y. Sun, M. Peng, and S. Mao, "Deep reinforcement learning-based mode selection and resource management for green fog radio access networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1960–1971, 2019.

[18] G. Sun, H. Al-Ward, G. O. Boateng, and G. Liu, "Autonomous cache resource slicing and content placement at virtualized mobile edge network," *IEEE Access*, vol. 7, pp. 84727–84743, 2019.

[19] R. Zhao, X. Wang, J. Xia, and L. Fan, "Deep reinforcement learning based mobile edge computing for intelligent Internet of Things," *Physical Communication*, vol. 43, article 101184, 2020.

[20] F. Jiang, Z. Yuan, C. Sun, and J. Wang, "Deep Q-learning-based content caching with update strategy for fog radio access networks," *IEEE Access*, vol. 7, pp. 97505–97514, 2019.

[21] C. Wang, W. Li, D. Li, M. Song, C. Dong, and X. Wang, "Edge caching via content offloading in heterogeneous mobile opportunistic networks," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 787–794, Singapore, Singapore, December 2018.

[22] N. Wang, G. Shen, S. K. Bose, and W. Shao, "Zone-based cooperative content caching and delivery for radio access network with mobile edge computing," *IEEE Access*, vol. 7, pp. 4031–4044, 2019.

[23] X. He, K. Wang, H. Huang, T. Miyazaki, Y. Wang, and S. Guo, "Green resource allocation based on deep reinforcement

learning in content-centric IoT," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 3, pp. 781–796, 2020.

[24] N. C. Luong, D. T. Hoang, S. Gong et al., "Applications of deep reinforcement learning in communications and networking: a survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[25] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2018.

[26] F. Xu, F. Yang, S. Bao, and C. Zhao, "DQN inspired joint computing and caching resource allocation approach for software defined information-centric Internet of Things network," *IEEE Access*, vol. 7, pp. 61987–61996, 2019.