Multi-agent Deep Reinforcement Learning-Based Content Caching in Cache-Enabled Networks

Sai Liu[1], Jia Chen[1], Deyun Gao[1](✉), Meiyi Yang[1], and Junfeng Ma[2]

[1] National Engineering Research Center of Advanced Network Technologies, School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
{21120089,chenjia,gaody,19111045}@bjtu.edu.cn

[2] China Academy of Information and Communication Technology, Beijing 100191, China
majunfeng@caict.ac.cn

Abstract. Deploying popular contents during the off-peak time via node cooperative caching has been proved to be an important measure to alleviate the link burden of wired multi-hop network. However, due to the unknown and time-varying content popularity in practice, it is imminent to optimize the content placement to effectively utilize the limited caching storage. In this paper, to address these issues, we investigate a multi-agent reinforcement learning (DRL) mechanism to intelligently deploy contents in dynamic environments. The optimization of content caching is modeled as a cooperative Markov decision process (MDP) to minimize the content transmission cost in the network. Then, a multi-agent deep deterministic policy gradient-based collaborative caching algorithm (MADDPG-CC) is proposed to solve this optimization problem with the goal of maximizing long-term caching reward. Extensive simulations have demonstrated the superior performance of the proposed algorithm in reducing the content transmission cost compared to existing caching algorithms, as well as its compatibility with variable and dynamic network environments.

Keywords: Collaborative caching · Content transmission cost · Markov decision process · Multi-agent deep deterministic policy gradient

1 Introduction

In recent years, with the boom in new multimedia applications such as short-form video applications, a huge volume of content that requires additional resources has been generated on the network, and this leads to the exceptional growth of network traffic. In order to reduce the burden of content transmit in the

Supported by School of Electronic and Information Engineering, Beijing Jiaotong University.

network and improve the users' service experience, mobile edge caching (MEC) has been proposed in many studies, such as [1–3]. That is, a limited local cache space is allocated in the edge node to store some popular requested content, and when the user requests for this popular content, the edge node can respond directly to the user's content request. However, as the number of users of these applications proliferates, it is difficult to fully satisfy user demand for popular content by considering caching deployments solely in the edge nodes. And now resource storage and caching technologies are also pervading various network architectures such as content delivery networks (CDNs) and information centre networks (ICNs). So, inspired by this, we consider a wired multi-hop network that can support caching, and the central issue in this type of network scenario is how to make full use of the limited local cache space of the cache nodes to minimize the content transmission cost in the network.

Efficient utilization of limited node cache resources is highly dependent on the content popularity. However, it is dynamic in both time and space, which is a priori unknown. To this end, machine learning is employed to predict content popularity based on historical observations and then optimize content placement, such as [4–6]. In [4], the authors propose a transfer learning-based caching scheme to predict trends in the popularity of user-requested content by extracting rich contextual information from the target and source domains, while the authors in [5] and [6] used multi-armed bandit (MAB) to predict the content popularity. However, the effectiveness of these efforts depends heavily on the accuracy of the predictions, so there is a great deal of uncertainty and ultimately it is difficult to obtain an optimal caching strategy.

The rise and widespread use of reinforcement learning has shown promise in solving the caching decision problem for mobile edge nodes in wireless networks. It adapts to the dynamics of the network environment without the need for prior knowledge of the dynamics of the network environment. To deal with higher dimensional state spaces, deep reinforcement learning (DRL) has started to be used in recent years of research to develop caching strategies. Many of the current studies related to MEC use centralized deep reinforcement learning, where edge nodes in a static network environment are trained centrally to obtain the full caching policy directly, as in [7,8]. However, for the complex network environment and the dynamic content popularity, applying centralized deep reinforcement learning algorithms to this is hardly successful. Then some of the latest research for MEC is starting to use multi-agent deep reinforcement learning algorithms, such as [9–11], in which all agents can not only fully perceive and acquire the state of the complex and changing network environment through distributed learning, but also better collaborative caching between agents through centralized training. However, in all of these studies, the authors only considered the collaborative nature of adjacent edge nodes, which is very different from caching in wired multi-hop networks.

In a wired multi-hop network, the source node generating the content request is directly connected to the network, so there is an interaction between the cache update policy of the source node and the cache policies of the intermediate nodes on the routing path. That is, these cache nodes are all in a cooperative relationship with each other and are not limited to neighboring nodes. So when considering caching in a wired multi-hop network, we need to take full account of the collaboration between all the cache nodes in the network, which also dictates that the edge node caching strategy for wireless networks cannot be directly applied to wired multi-hop networks.

In this paper, we aim to optimize content caching among cooperative nodes by minimizing the expected total transmission cost over a finite time horizon in cache-enabled wired networks without knowing content popularity in advance. In our experiments, we use the reduction in transmission cost as the caching reward compared to caching without requesting content at the cache node, and then we reformulate this content caching as a Markov decision process (MDP) to maximize the total caching reward. In addition, we take into account the complexity network environment and the collaboration between nodes, and choose to use a multi-agent deep reinforcement learning algorithm to train and learn the caching policy. The main contributions of this paper are as follows.

1) Analyzes the problem to optimizing content caching policies in cache-enabled wired multi-hop networks and transmits it as a cooperative MDP by defining the caching reward.
2) Considering the complex and variable nature of the actual network environment and the cooperative relationship between caching nodes, we propose a multi-agent-based intelligent caching algorithm MADDPG-CC.
3) Through extensive simulations, we demonstrate that our algorithm converges up to nearly 10% better and significantly faster in the same network environment compared to various other algorithms. In addition, our algorithms are highly compatible with complex and changing network environments.

2 System Model

2.1 Model Building

When considering a practical network model, we choose to use directed graphs to represent general multi-hop networks, whose general expression is $G = (N, L)$, where N denotes the set of all nodes in this multi-hop network, and L denotes the set of all links between any two adjacent nodes, i.e., $\forall l = (n_1, n_2) \in L, N_s \ll N_c$. In addition, we assume that a total of M different content (e.g., video, audio, image, text, etc.) is stored in the content library of this multi-hop network. The content base of the entire network is denoted by E, i.e., $E = \{e_1, e_2, ..., e_m, ..., e_M\}$, and in the model, we set the size of a content to 1 and represent the different contents with different natural numbers. The nodes in this network are divided into two types, server nodes and cache nodes, whose sets are denoted as N_s and N_c respectively, and $N = N_s \cup N_c$, $N_s \ll N_c$. Each cache node i has a local cache space D_i of size V, i.e., $D_i = \left\{d_1^i, d_2^i, ..., d_v^i, ..., d_V^i\right\}$, $V \ll M$. And each server node s has a local cache space that is larger than the cache node. For each content e_m in the network content library E, there exists a server node s that stores it uniquely and fixedly, i.e., the content in the entire network content library is stored non-repeatedly in each server node in the network.

In addition, we divide a longer period of time T into different time slot t, i.e., $T = \{1, 2, ..., t\}$. We assume that all cache nodes i, except server node s, receive a certain number of content requests at each time slot t. We denote the set of content requests from cache node i at time slot t as $b_i(t)$, i.e., $b_i(t) = \left\{b^t_{i,1}, b^t_{i,2}, ..., b^t_{i,w}, ..., b^t_{i,W}\right\}$, where $b^t_{i,w}$ indicates the specific content of the request, while W is the total number of all content requests received by cache node i at time slot t. To ensure that the number of content requests received by each cache node i within each time slot better matches a realistic scenario, we set the number of content requests W received by each cache node i within each time slot to follow an independent Poisson distribution process with arrival rate λ_i. In addition, for the request contents received by each cache node i at each time slot t, we set it to follow $Zipf$ distributions with parameter α.

As mentioned earlier, for each cache node i there is a finite size of local cache space D_i. In this paper, we define the set of local cache contents maintained by all cache nodes in time slot t as the cache matrix $x\left(t\right)$, and $x\left(t\right) = \left\{x_i\left(t\right), \forall i \in N_c\right\}$, where $x_i\left(t\right)$ represents the vector consisting of the local cache contents maintained by cache node i in time slot t. It is important to note here that the total amount of content stored in the local cache space per time slot t for each cache node i must not exceed the maximal cache space V for that node, and that both the content stored in the local cache space for each cache node i and the requested content from each cache node i should be available for matching within the server nodes in the network.

In network G, each link is assigned a fixed weight as the data transmission cost of that link, after taking into account the various types of transmission costs of the link (e.g., transmission delay, energy consumption, traffic, etc.). In this paper, we set cache nodes to communicate with server nodes according to the least weighted path. So the request $b_{i,m}$ for content m from cache node i complete the transmission of the content request and response with the target server node s, where content m is stored, via the least weighted path $p^m_{i,s}$. In addition, it is also important to note that the cost of transmitting the content request itself across the network is minimal compared to the content response. Therefore, in this paper, we ignore the request transmission cost. Moreover, during the transmission of request $b_{i,m}$ along the path $p^m_{i,s}$, if the intermediate cache node j passes through a successful match for content m, the transmission is terminated and cache node j follows the reverse symmetric path, not necessarily the path of least weight of cache nodes i and j, and responds directly to the request $b_{i,m}$ from the cache node i.

2.2 Problem Transformation

Based on the main idea of deep reinforcement learning, we transmit the problem to optimizing content caching policies in cache-enabled wired multi-hop networks into a MDP, in which each cache node acts as an agent and the cache-enabled network acts as environment. Define the each critical elements of MDP as follows:

State: The state of each agent reflects the local network environment. So, when we define the state of each agent, it should include the storage state of the agent's local cache space and the state of content requests received by the agent. In turn, the global state can be defined as $s\left(t\right) = \left\{x\left(t\right), b\left(t\right)\right\}$, $x\left(t\right) = \left\{x_{i,m}\left(t\right), \forall i \in N_c, \forall m \in E\right\}$ denotes the vector consisting of the contents stored in the local cache space of agent i at time slot t, and $b(t) = \{b_{i,m}(t), \forall i \in N_c, \forall m \in E\}$ denotes the vector of all request contents received by agent i at time slot t. So the state of agent i at time slot t can then be expressed as $s_i\left(t\right) = \left\{x_i\left(t\right), b_i\left(t\right), \forall i \in N_c\right\}$, where $b_i\left(t\right)$ is a vector of dimension M (M denotes the size of the content library in this network), and the index corresponding to each position in the vector indicates the specific content requested. The value corresponding to the index at each position indicates the number of times agent i requests that content at time slot t. Our goal is to continuously train the neural network at time slot $t+1$ using the global state $s\left(u\right)$ of some historical time slots, $u \in \{1, 2, 3, ..., t\}$, with the rewards obtained from these time slots, and then learn to obtain the optimal caching policy for each agent within time slot $t+1$.

Action: At the end of time slot t, agent i gets its state vector $x_i\left(t\right)$ as input to its actor network, and the output from the actor network is the caching policy $p_i\left(t\right)$ of agent i, where $p_i\left(t\right)$ is a vector of dimension M, and the specific value in the vector represents the value of the indexed content of the corresponding location of the cache selected by agent i, i.e., the larger the value, the bigger reward will be given for caching its corresponding indexed content. We then note the value of the first V index positions of the value according to the cache policy $p_i\left(t\right)$ as 1, and the rest of the index positions as 0. Thus we can obtain the cache action vector $a_i\left(t\right)$ consisting of 0, 1 binary numbers, and the index corresponding to the vector position where the number 1 is located indicates the request content that agent i should cache updates. So the global action matrix at time slot t can be defined as $a\left(t\right) = \left\{a_i\left(t\right), \forall i \in N_c\right\}$.

Reward Function: At time slot t, we define the transmission cost spent by target server node s to respond to the agent i that sends the request $b_{i,m}\left(t\right)$ with content m via the path $p_{i,s}^m$ as $C_{i,s}^m$, and we define the transmission cost spent by cache node j on the path $p_{i,s}^m$ storing content m to respond directly to agent i via the reverse symmetric path as $C_{i,j}^m$. And the difference between the above two costs we define as the reward given to cache node j by caching content m. We define that the set of intermediate nodes contained in the path $p_{i,s}^m$ is $N_{i,p_{i,s}^m,s}$, $i, s \in N_{i,p_{i,s}^m,s}$, then $C_{i,s}^m$ can be expressed as:

$$C_{i,s}^m = \sum_{f,k \in N_{i,p_{i,s}^m,s}} W_{l_{f,k}} \tag{1}$$

where $W_{l_{f,k}}$ is the link weight corresponding to the link $l_{f,k}$ between node f and node k.

Likewise, let the set of intermediate nodes contained in the path $p_{i,j}^{m}$ between agent i and agent j be $N_{i,p_{i,s}^{m},j}$, $i,j \in N_{i,p_{i,s}^{m},j}$, then $C_{i,j}^{m}$ can be expressed as:

$$C_{i,j}^{m} = \sum_{f,k \in N_{i,p_{i,s}^{m},j}} W_{l_{f,k}} \tag{2}$$

We set $y_{i,j}^{m}\left(t\right)$ to be a binary judgment marker. If the request $b_{i,m}\left(t\right)$ from agent i to content m is the first to match successfully in the local cache space of agent j at time slot t, then $y_{i,j}^{m}\left(t\right)$ set to 1, otherwise, set to 0. Then the total reward $r_j\left(t\right)$ obtained by agent j at time slot t can be expressed as:

$$r_j\left(t\right) = y_{i,j}^{m}\left(t\right) \sum_{i \in N_c} \sum_{m \in b_i(t)} \left(C_{i,s}^{m} - C_{i,j}^{m}\right) \tag{3}$$

To reduce the order of magnitude of the reward value obtained by each agent in each time slot, we use the reduction $\overline{r}_i\left(t\right)$ in the average transmission cost of the response to each content request in agent$i$ at time slot t, as the reward obtained by agent i in that time slot as a result of caching content $x_i\left(t\right)$, i.e., $\overline{r}_i\left(t\right) = r_i\left(t\right)/W$. Thus, we can obtain the vector set of rewards for all agents at time slot t as $r\left(t\right) = \{\overline{r}_i\left(t\right), i \in N_c\}$.

2.3 MADDPG-Based Cooperative Caching Algorithm

The MADDPG [12] algorithm is an Actor-Critic based algorithm that uses a centralized training and decentralized execution architecture. As can be seen in Fig. 1, it is an architecture consisting of multiple actor-critic. During training, each agent's actor network obtains local observation, while the agent's critic network obtains global observation (i.e., the local observation and actions of other agents) for centralized global training. Once the model is trained, it only needs the agent to interact with the environment to obtain the optimal action decision. The MADDPG algorithm has three main features: (i) the agents are centralized training, decentralized execution; (ii) the dynamical model of the environment does not need to be known; and (iii) the cooperative or competitive environment is applicable.

In this subsection we propose a cooperative caching algorithm based on MADDPG, called MADDPG-CC. The algorithm treats each cache node in the network as a separate agent, and the content deployment policy of each cache node is trained collaboratively via the continuous interaction between all agents and the network environment, to minimize the transmission cost. By defining the input states, output actions and reward functions for each agent in the previous sub-section, we give the specific training learning process of the MADDPG-CC algorithm in interaction with the network environment in the form of pseudo-code, see Algorithm 1.

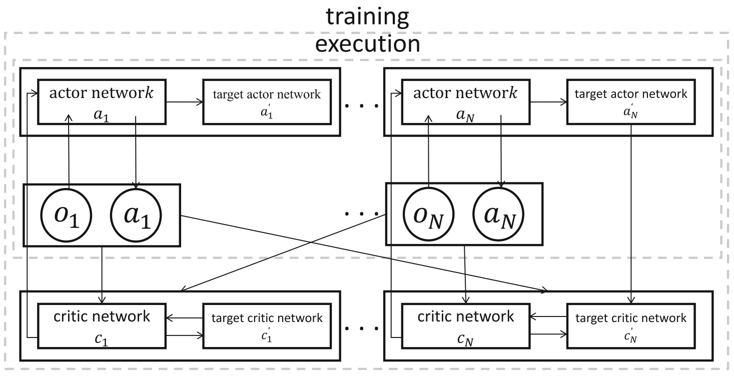

Fig. 1. MADDPG algorithm framework.

Algorithm 1. MADDPG-CC

```
Input: Initialize online parameters of actor network and online parameters of critic
  network for each agent; Initialize the local cache contents of all server nodes in the
  network.
Output: Average Reward
  for episode=1 to max-episode-number do
      Initialize the global local cache matrix $x(0)$;
      Initialize the global content request matrix $b(0)$;
      Observe the global state matrix $s(0)$
      for $t$=1 to max-episode-length do
          For each agent $i$, observe a policy $p_i\,(t)$ by current state $s_i\,(t)$;
          For each agent $i$, execute action $a_i\,(t)$ with respect to policy $p_i\,(t)$;
          Observe new the global local cache matrix $x(i+1)$;
          For each agent $i$ obtains request vector $b_i\,(t+1)$;
          Observe the global reward $r\,(t)$;
          Obtain the global state matrix $s(t+1)$;
          $s^{'} \leftarrow s(t+1)$;
          Store $\{s(t), a(t), r(t), s^{'}\}$ in replay buffer $D$;
          $s(t) \leftarrow s(t+1)$;
      end for
      for agent $i$=1 to $N_c$ do
          Sample a random mini batch of S samples $\{s^k, a^k, r^k, s^{'k}\}$ from $D$;
          Set target network;
          Minimizing the loss and update the critic network;
          Update the actor network;
      end for
      Update target network parameters for each agent $i$;
  end for
```

3 Simulation Experiments

In this section, we conduct extensive simulation experiments to verify the performance of the MADDPG-CC algorithm in various complex and dynamic network environments.

3.1 Experimental Simulation Environment

Building the network environment required for our simulation experiments, we choose to use the US Abilene Network. In this network topology, 11 nodes and 14 links are included. In the simulation experiments, we choose three of the nodes as server nodes, each of which stores one-third of the content of the network content library, and the remaining eight nodes as cache nodes, each of which acts as an independent agent. Each cache node follows an independent Poisson distribution process to generate the number of content requests for that cache node. Unless otherwise specified, we set the simulation parameters as follows: the arrival rate of the Poisson distribution $\lambda = 100$, the network content library size $M = 120$, the local cache space size of each cache node $V = 15$ and $Zipf$ distribution with the parameter $\alpha = 0.8$. In addition, each link is assigned a weight derived from TOTEM project [13]. Finally, some key parameters in the implementation of our algorithm are given in Table 1.

Table 1. Parameters for algorithm.

Parameters	Values
Agent number	8
Size of replay buffer	1000000
Size of mini-batch	1024
Learning rate	0.001
Discount	0.95
Maximum number of episodes	2500
Number of slots in each episode	50

3.2 Experimental Results

In this section, we analyze our proposed algorithm from two main aspects, the training process of MADDPG-CC in different network environments and the performance comparison between different algorithms, to demonstrate its superiority in performance.

Training Process in Different Network Environments. First, we verify the compatibility of our algorithm in various complex and dynamic network environments by varying the following variables in the network environment to obtain different convergence results.

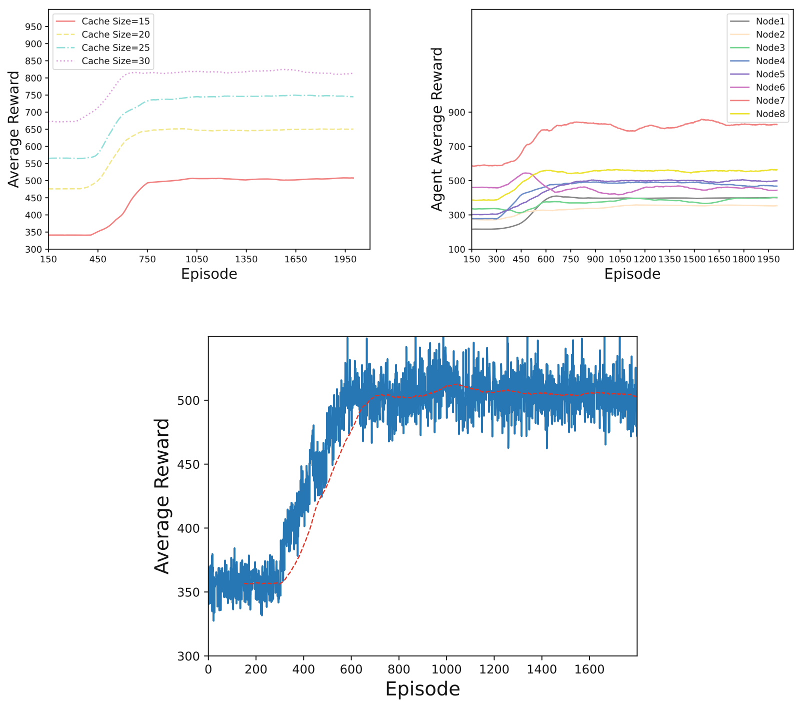

Fig. 2. Convergence curves of MADDPG-CC in different network environments: (a) Different local cache space sizes ($V = 15, 20, 25, 30$). (b) Different *Zipf* distribution parameters for different cache nodes ($\alpha = 0.2, 0.4, 0, 6, 0.8, 1.0, 1.2, 1.4, 1.6$). (c) Alternating *Zipf* distribution parameters ($\alpha = 0.8$ or 1.6).

The Size of Local Cache Space for Cache Nodes. Figure 2a shows the training curves of our algorithm for cache nodes with local cache space sizes of 15, 20, 25 and 30, and the results show that our algorithm still has good convergence results in these network environments. Moreover, as the local cache space of the cache node increases, the hit rate of the requested content at the intermediate cache nodes also increases, so the final convergence result obtained also increases.

Zipf Distributions of Cache Nodes. Figure 2b shows the convergence of the average reward value obtained by each cache node when the parameters of *Zipf* distributions followed by our algorithm in generating the requested content of different cache nodes are set to different values (cache node 1 to cache node 8 are set to 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6 in that order), from which we can see that all the cache nodes also converge separately. The difference in the final convergence results of different cache nodes is due to their specific positions in the network topology and the different link weights.

In addition, when conducting simulation experiments, we also simulated a network environment with dynamic and complex changes in the request content popularity of cache nodes by setting the parameter α of $Zipf$ distributions followed by the request content of cache nodes to 0.8 at odd episodes and 1.6 at even episodes, and the obtained training curves are shown in Fig. 2c, and the final convergence results demonstrate that the MADDPG-CC algorithm also shows good performance when training cache nodes with dynamically changing request content popularity, thus indicating that our algorithm is predictive of the changing trend of request content popularity.

Performance Comparison of Different Algorithms. To verify the performance difference between our proposed algorithm and other traditional caching algorithms as well as centralized reinforcement learning algorithms, we consider the following three algorithms chosen to compare with our proposed algorithm: least recently used (LRU), least frequently used (LFU) and deep deterministic policy gradient (DDPG).

The convergence of the average reward value curve obtained by the four algorithms is shown in Fig. 3. Since the training result data-set obtained from each training was too large, we considered taking an average every 150 episodes when plotting, and then plotted the convergence curve based on the obtained average value, so the starting value of the number of episodes in the horizontal coordinate is 150 instead of 0. In addition, for the initial 400 episodes, the agent only interacts with the network environment, completes online learning and generates experience bars, but does not update the actor and critic networks, so we can see from the figure that the average reward obtained by the two algorithms, MADDPG-CC and DDPG, do not change during this phase. After 400 episodes, the agent starts to update the parameters of the neural network by extracting experience entries from the replay buffer, so in the interval from 400 to 750, the result curves of the two algorithms start to rise significantly, and it can be seen that the MADDPG-CC algorithm converges significantly faster than DDPG. After 750 episodes, the results of both algorithms slowly converge, and from the final convergence results we can also see that the convergence results obtained by the two deep reinforcement learning algorithms are much greater than those obtained by the two traditional algorithms, LRU and LFU. And the final convergence value obtained by the MADDPG-CC algorithm is also slightly greater than that obtained by DDPG. We believe that the reason why the final convergence results of the two deep reinforcement learning algorithms are relatively close is that the size of the network content library set in the simulated experimental environment is small, so the advantages of our algorithm are not reflected.

In order to verify our above idea, we adjusted the size of the network content library in the simulation environment from 120 to 240 and 600 respectively, with other parameters unchanged, and re-trained the two algorithms with MADDPG-CC and DDPG, and the results are shown in Fig. 4. When the size of the content library is adjusted to 240 and 600, the difference between the convergence results

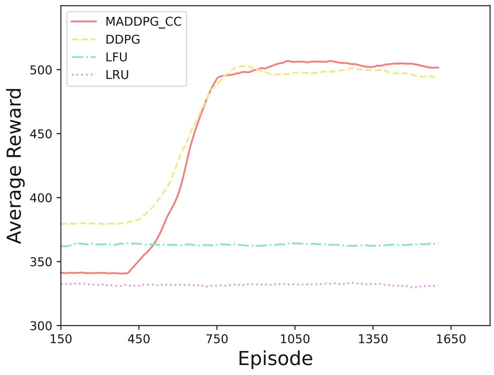

Fig. 3. Convergence curves for the four algorithms.

of the two algorithms is more obvious. Moreover, we can clearly see from all three figures that the convergence speed of the MADDPG-CC algorithm is also significantly faster than that of DDPG.

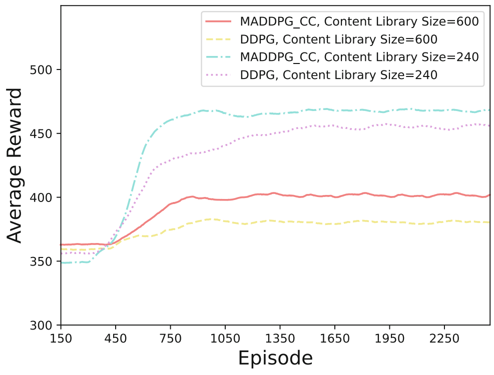

Fig. 4. Convergence curves of MADDPG-CC and DDPG at content library sizes of 240 and 600.

4 Conclusion

In this paper, we address the problem of content caching in wired multi-hop networks with the aim of minimizing long-term content transmission cost. Due to the dynamic and random nature of the network environment, we remodel this caching decisions making problem as a MDP and address it by learning caching policy directly online based on the proposed MADDPG-CC framework. Based on extensive simulation results, we demonstrate that our proposed algorithm is well compatible with various dynamically changing network environments.

Furthermore, our experimental results demonstrate the superior performance compared to current existing algorithms in different network environments.

Acknowledgements. This work is supported by the National Key Research and Development Program of China (grant no. 2018YFE0206800), the fundamental research funds for the central universities (no. 2021YJS013), and the National Natural Science Foundation of China (grant no. 61971028).

References

1. Yao, J., Han, T., Ansari, N.: On mobile edge caching. IEEE Commun. Surv. Tutor. **21**(3), 2525–2553 (2019)
2. Qiu, L., Cao, G.: Popularity-aware caching increases the capacity of wireless networks. IEEE Trans. Mob. Comput. **19**(1), 173–187 (2019)
3. Wang, R., Zhang, J., Song, S., Letaief, K.B.: Mobility-aware caching in D2D networks. IEEE Trans. Wirel. Commun. **16**(8), 5001–5015 (2017)
4. Hou, T., Feng, G., Qin, S., Jiang, W., Letaief: Proactive content caching by exploiting transfer learning for mobile edge computing. IEEE Trans. Wirel. Commun., 1–6 (2017). Proc. GLOBECOM
5. Song, J., Sheng, M., Quek, T.Q.S.: Learning-based content caching and sharing for wireless networks. IEEE Trans. Commun. **65**(10), 4309–4324 (2017)
6. Blasco, P., Gndz, D.: Learning-based optimization of cache content in a small cell base station. In: Proceedings of the ICC, pp. 1897–1903, June 2014
7. Zhong, C., Gursoy, M.C., Velipasalar, S.: A deep reinforcement learning-based framework for content caching. In: Proceedings of the CISS, pp. 1–6, March 2018
8. Zhu, H., Cao, Y., Wang, W., Jiang, T., Jin, S.: Deep reinforcement learning for mobile edge caching: review, new features, and open issues. IEEE Netw. **32**(6), 50–57 (2018)
9. Jiang, W., Feng, G., Qin, S., et al.: Multi-agent reinforcement learning for efficient content caching in mobile D2D networks. IEEE Trans. Wirel. Commun. **18**(3), 1610–1622 (2019)
10. Feriani, A., Hossain, E.: Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: a tutorial. IEEE Commun. Surv. Tutor. **23**, 1226–1252 (2021)
11. Wang, F., Wang, F., Liu, J., Shea, R., Sun, L.: Intelligent video caching at network edge: a multi-agent deep reinforcement learning approach. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 2499–2508. IEEE (2020)
12. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments (2017). arXiv preprint arXiv:1706.02275
13. TOTEM progect. https://www.cs.utexas.edu/yzhang/research/AbileneTM/. Accessed 4 Oct 2003