A Multi-Armed Bandits Learning-Based Approach to Service Caching in Edge Computing Environment

Jinpeng Li[1], Jiale Zhao[1](✉), Peng Chen[2], Yunni Xia[1](✉), Fan Li[3], Yin Li[4], Feng Zeng[5], and Hui Liu[6]

[1] School of Computer, Chongqing University, Chongqing 400030, China
zhaojiale0415@163.com, xiayunni@hotmail.com
[2] School of Computer and Software Engineering, Xihua University, Chengdu 610039, China
[3] Key Laboratory of Fundamental Synthetic Vision Graphics and Image Science for National Defense, Sichuan University, Chengdu 610065, China
[4] Guangzhou Institute of Software Application Technology, Guangzhou 510990, China
[5] Discovery Technology (Shenzhen)Limited, Shenzhen 518129, China
[6] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100083, China

Abstract. Mobile edge computing (MEC) is a newly emerging concept that provides significant local computing power and reduces end-to-end latency. In MEC environments, caching frequently accessed services on edge servers effectively reduces latency and improves system responsiveness. An ongoing research topic in such a cachable MEC context is to design novel algorithms for yielding high-quality caching decision that guarantee high user-perceived quality-of-service (QoS) and high system responsiveness of delivery of cached content with the difference of caching capacities of edge servers and diversified content popularity appropriately addressed. In this article, we propose a multi-armed bandits learning-based method busing a Thompson sampling for generating caching decisions. We introduce a genetic multi-armed bandits algorithm (GMAB), which synthesizes the genetic algorithm (GA) and multi-armed bandits (MAB), for optimizing caching effectiveness with timing and space constraints. The experiment results show that GMAB outperforms traditional methods in terms of multiple aspects.

Keywords: Edge computing · Multi-armed bandits learning · Service caching · Thompson sampling · Genetic algorithm

1 Introduction

The world today is witnessing a rapid growth in the number of smart Internet-of-Things (IoT) devices including mobile phones, smart watches, mobile computers and autonomous cars [1]. These devices introduce massive data computing requirements, especially for delay-sensitive services, which brings great challenges to the mobile cloud computing (MCC). In a traditional centralized environment, computation-demanding tasks are offloaded to the resource-rich cloud center, resulting in significant contents transmission delays for sending and receiving data from IoT to the cloud, which seriously affects efficiency of content delivery. MEC enables intelligent content caching at the network edge to reduce traffic and enhance content delivery efficiency. In MEC architecture, popular content can be deployed at the MEC server to improve users' quality of experience (QoE). The edge servers with a certain cache of computing resources are attached to base stations or access points close to users, providing users with real-time computing and data storage services at the edge of network. In the MEC environment, tasks can be offloaded from IoT devices to the optimal edge servers to avoid data transmission from the cloud.

Despite its benefits, caching at MEC faces several challenges, such as unpredictable user mobility, limited storage space of edge nodes, and unpredictability of content requests. One critical challenge is to achieve good tradeoffs among efficiency of content delivery (in terms of, e.g., cache hit rate and response time) and system overhead (in terms of, e.g., backhaul traffic).

In this study, we propose a genetic multi-armed bandits (GMAB) learning-based method to yielding dynamic service caching schemes in MEC, the main contributions are as follows:

1) We design a decentralized decision-making mechanism, where each edge server runs a GMAB in a decentralized manner.
2) We accelerate convergence speed by leveraging a genetic reinforcement learning algorithm, which is capable of adapting caching schemes to real-time changes of task types.
3) To evaluate the performance of GMAB, we conduct extensive simulations by using a real-world data set and show that our proposed method outperforms its peers in terms of multiple performance indicators.

This paper is organized as follows: In Sect. 2, we review some related work. The system models and problem definition are given in Sect. 3. The proposed method is described in Sect. 4, and performance evaluation is presented in Sect. 5.

2 Related Work

Caching data on the edge server, which is near the user, is a evolving technique used in various areas. In recent years, it has attracted many researchers' attention as an active research topic.

Least Recently Used (LRU) and Least Recently Visited (LFU) are the two most used strategies in this direction [2,3]. Xia *et al.* [4] provided an online algorithm for the collaborative problem in MEC environment by modeling the edge caching problem as a constrained optimization matter. Zhao *et al.* [5] introduced a service placement method over vehicular to address the challenge of improving the caching hit rate. Zeng *et al.* [6] integrated user behavioral preferences and contextual zoom to develop a heuristic intelligent caching scheme. The objective is to optimize the priority content under the historical request count of the corresponding content.

Recently, machine learning models and algorithms showed their ability in yielding and optimizing MEC caching schemes. Sengupta *et al.* [7] addressed the distributed caching problem in mobile edge networks from the perspective of reinforcement learning. Zhong *et al.* [8] proposed an actor-critic (AC) learning framework to optimize content delivery latency. Song *et al.* [9] studied the collaborative shared cache problem of static users in the MEC environment, modeled a single agent learning mechanism for optimizing caching decisions. Wu *et al.* [10] formulate the resource allocation strategy as a joint optimization problem, and they use deep Q-learning network (DQN) for solving this problem. Recently, the shortcoming of DQN has also gained considerable deliberation because of the difficulties in handing a large action space. Qiao *et al.* [11] proposed a cooperative edge caching scheme suitable for complicated action space relying on a deep deterministic policy gradient (DDPG) model.

Multi-armed bandits (MAB)-based methods is frequently seen in this direction as well. Malazi *et al.* [12] proposed a distributed caching strategy that describes the edge service placement problem as a multi-armed bandit problem and used the upper confidence bound (UCB1) algorithm for optimization. Wu *et al.* [13] considered MEC caching mechanisms over multi-rat heterogeneous networks and extracted the parts that can be measured in real-time when they used UCB1 to calculate rewards for reducing the workload. Chen *et al.* [14] studied the spatial-temporal edge service caching problem of an application service providers under a limited budget and proposed an contextual bandit learning algorithm to optimize the edge computing performance. Xu *et al.* [15] proposed a collaborative cache management algorithm that maximizes cache service traffic while minimizing bandwidth costs. Jiang *et al.* [16] studied a collaboration scheme between MEC servers to optimize content caching and delivery performance between MEC and mobile devices. Ren *et al.* [17,18] proposed a grouping-based caching strategy and considered allocating storage resources to reduce the average latency and total energy consumption in content retrieval.

3 System Models and Problem Formulation

3.1 System Model

In this article, we consider a MEC environment built upon a remote cloud and a set of base stations equipped with edge servers linked to the cloud through a backhaul network. As shown in Fig. 1, our caching model is built upon the MEC environment and it comprises a task offloading model and a service cache model. When a task request reaches the edge server, it can only be executed if the corresponding type of service is cached there. Our aim is thus to optimize the schedules for updating the service cache with satisfactory performance.

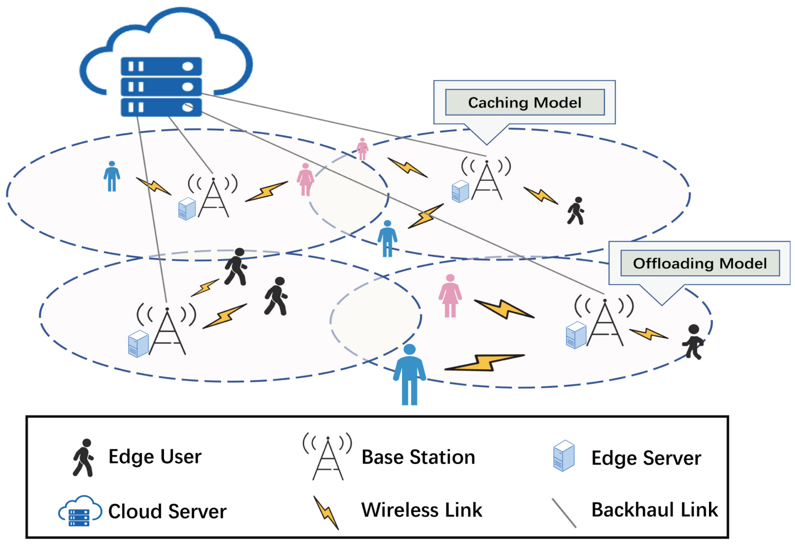

Fig. 1. Edge computing system model.

The central cloud is capable of handling all types of tasks and c_c^i is the capacity for service s_i in the cloud. $E = \{e_1, e_2, ..., e_m\}$ denotes the set of edge servers deployed in the current area. Each edge server can be described by a 5-tuple $e_i = (L_i, r_i, b_i, C_i, N_i)$, where $L_i = (lot, lat)$ denotes the geographical position of e_i, r_i the radius of its signal coverage, b_i the capacity of caching services for e_i, $C_i = \{c_i^1, c_i^2, ..., c_i^n\}$ a set of the ability to handle tasks, c_i^j the computational capacity of edge server e_i for task k_j and $N_i = \{e_{i_1}, e_{i_2}, ..., e_{i_s}\}$ the set of neighboring and wireless network-reachable servers of e_i. $S = \{s_1, s_2, ..., s_k\}$ is defined as the set of all services. $K(t) = \{k_1, k_2, ..., k_n\}$ denotes the set of all tasks for e_i at time t. Each task can be described by a 4-tuple $k_j = (q_j, s_l, d_j, p_j)$, where q_j denotes the location of task k_j, s_l the required service type by task k_j, d_j the up-link data size and p_j the computational overhead. A user and a base stations are connected with Standalone (SA) or Non-Standalone Access (NSA) 5G network. Base stations themselves are inter-connected via X2 or Xn links [19]. $W_e(t)$ denotes the bandwidth of 5G at time t, and $W_c(t)$ the bandwidth of the inter edge-cloud backbone network at time t. All symbols appearing in this paper are shown in Table 1.

Table 1. Notion table

Variable	Description
c_c^i	The capacity for handling s_i in the cloud
E	A set of edge servers
e_i	The i-th edge server
L_i	The geographical position of e_i
r_i	The signal radius of e_i
b_i	The capacity of caching services for e_i
C_i	A set of the ability of e_i to handle tasks
N_i	A set of neighboring and wireless network-reachable servers of e_i
S	A set of all services
$K(t)$	A set of all tasks for e_i at time t
k_j	The j-th task in K^t
q_j	The location of k_j
s_l	The required service by k_j
d_j	The up-link data size of k_j
p_j	The computational overhead of k_j
$W_e(t)$	The bandwidth of 5G at time t
$W_c(t)$	The bandwidth of the inter edge-cloud backbone network at time t
$y_{i,j}(t)$	A binary variable showing whether the service sj is cached in e_i at time t
$q(s_j)$	The data size of s_j
$tcor_{i,j}(t)$	The transmission time when k_j is offloaded to e_i and no malfunctioning
$terr_{i,j}(t)$	The transmission time when k_j is offloaded to e_i and failure occurs
$tt_{i,j}(t,u)$	The transmission time between k_j to e_i
$tc_{i,j}(t)$	The calculation delay of the k_j
$ra_i(t)$	The average hit rate of e_i

3.2 Service Caching Model

Due to limited storage and computing capacity, edge servers can only cache a portion of services and thus cached content is updatable and replaceable. A binary variable $y_{i,j}(t)$ denotes whether the service s_j is cached in e_i at time t according to (1).

$$y_{i,t}(t) = \begin{cases} 1, \text{if } s_j \text{ is cached in } e_i \\ 0, \text{otherwise} \end{cases} \tag{1}$$

Instead of using traditional virtual machines, we consider that the caching system is empowered by containerization technologies (e.g., Docker), which can reduce cache costs and improve service utilization by enabling quick adaptation to request pattern variability. The maximum number of storage services for edge servers is limited by the container size according to (2). (2) implies that a service s_d can be cached in e_i at time t when the corresponding container size is sufficient.

$$\sum_{j=1,j\neq d}^{k}(y_{i,j}(t)\cdot q(s_j))+q(s_d)\leq b_i \tag{2}$$

where $q(s_j)$ denotes the data size of s_j.

3.3 Task Offloading Model

Task offloading model specifies how tasks are allocated for requesting cached content nearby. Each user can only be connected to one edge server e_i at a time. When receiving a request k_j, which requires service s_l to be processed, edge server e_i checks the availability of requested content. If e_i fails to hit s_l, this request is forwarded to the neighboring server N_i. When both e_i and N_i fail to offer s_i, the request is forwarded to the cloud. In addition, we assume that content delivery can fail, e.g., unstable wireless connection. In case of failure, e_i forwards k_j to the cloud as well. (3) specifies the transmission delay under different conditions when k_j is offloaded to e_i and the edge server is not malfunctioning.

$$tcor_{i,j}(t)=\begin{cases}\frac{d_j}{W_e(t)\log_2(1+\frac{\rho_j\tau_{ij}}{\lambda^2})}, & (y_{i,l}(t)=1)\\ \frac{2d_j}{W_e(t)\log_2(1+\frac{\rho_j\tau_{N_ij}}{\lambda^2})}, & (y_{i,l}(t)=0)\&(\exists y_{Ni,l}(t)=1)\\ \frac{d_j}{W_e(t)\log_2(1+\frac{\rho_j\tau_{cj}}{\lambda^2})}, & \text{otherwise}\end{cases} \tag{3}$$

where ρ_j denotes the transmission power of user devices, τ_{ij} the channel gain between the device requesting k_j and e_i, τ_{cj} the channel gain between the device requesting k_j and *cloud* and λ^2 the background noise power. If failure occurs, e_i forwards k_j to the cloud and the transmission delay of e_i is:

$$terr_{i,j}(t)=\begin{cases}tcor_{i,j}(t)+\frac{d_j}{W_c(t)\log_2(1+\frac{\rho_j\tau_{cj}}{\lambda^2})}, & (y_{i,l}(t)=1)\\ tcor_{i,j}(t)+\frac{d_j}{W_c(t)\log_2(1+\frac{\rho_j\tau_{N_{cj}}}{\lambda^2})}, & (y_{i,l}(t)=0)\&(\exists y_{Ni,l}(t)=1)\\ tcor_{i,j}(t), & \text{otherwise}\end{cases} \tag{4}$$

Therefore, the transmission time between user to e_i for offloading k_j at t is:

$$tt_{i,j}(t,u)=u\cdot terr_{i,j}(t)+(1-u)\cdot tcor_{i,j}(t) \tag{5}$$

where u is the error rate of edge servers, $terr_{i,j}(t)$ and $tcor_{i,j}(t)$ are defined in (3) and (4) respectively. The calculation delay of the k_j is represented by (6), and the average hit rate of e_i is defined as (7).

The calculation delay $tc_{i,j}(t)$ is determined by the computational overhead and the capacity for handling tasks, and the hit rate $ra_i(t)$ is the ratio of the number of tasks hit on the edge side to the total number of tasks.

$$tc_{i,j}(t)=\beta_{i,j}(t)\cdot\frac{p_j}{c_i^l}+(1-\beta_{i,j}(t))\cdot\frac{p_j}{c_c^l} \tag{6}$$

$$ra_i(t) = \frac{\sum_{j=1}^{n}(\beta_{i,j}(t) + \beta_{N_i,j}(t))}{n} \tag{7}$$

where $\beta_{i,j}(t)$ is a boolean indicator of whether k_j is executed at e_i, and n is the number of elements in $K(t)$, which represents the total number of tasks accessing e_i at time t.

3.4 Problem Formulation

In the MEC environment, edge servers can cache only a portion of services as mentioned earlier. Given the system model described above, our goals are as follows:

1. Minimize the average response delays according to (8).
2. Maximize hit rate according to (9).

$$\text{Min}: \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{t=0}^{t_{max}}(tt_{i,j}(t,u) + tc_{i,j}(t)) \tag{8}$$

$$\text{Max}: \sum_{i=1}^{m}\sum_{t=0}^{t_{max}} ra_i(t) \tag{9}$$

$$\begin{aligned} s.t. \quad &\textbf{C1.} \quad \sum_{j=1}^{k} y_{i,j}(t) < b_i \qquad \forall i, \forall t \\ &\textbf{C2.} \quad \sqrt{(q_j - Li)^2} < r_i \quad \forall i, \forall j, \forall t \end{aligned}$$

$C1$ indicates the size of the stored services is bounded by the caching size of the edge server. $C2$ represents the data transmission between edge servers and users occurs within the radius of the e_i signal coverage.

The above formulation can be considered as a capacitated facility location problem (CFLP) with a set of facilities (edge servers) and a set of customers (users). Facilities have a caching capacity, and their establishment process is similar to caching a service. Building a facility has the same cost as caching a service. The serving cost of customers includes both the service response time and the hit rate. The objective is to build facilities with constraints to minimize customer service costs. CFLP is NP-hard [20] and thus (8–9) is also an NP-hard as well.

4 The Proposed Method

This section outlines the genetic multi-armed bandits (GMAB) for yielding high-quality solutions to the formulation given above. The real-time service caching problem can be interpreted as a multi-armed bandits problem with n unknown and independent bonus probability distributions. Each edge server operates a multi-armed bandits algorithm where each arm represents a different service. Selecting an arm is equivalent to caching a service and an edge server periodically evaluates the current policy, as well as making caching replacements. An edge server selects multiple arms during each decision round based on the highest expected reward and the current caching state.

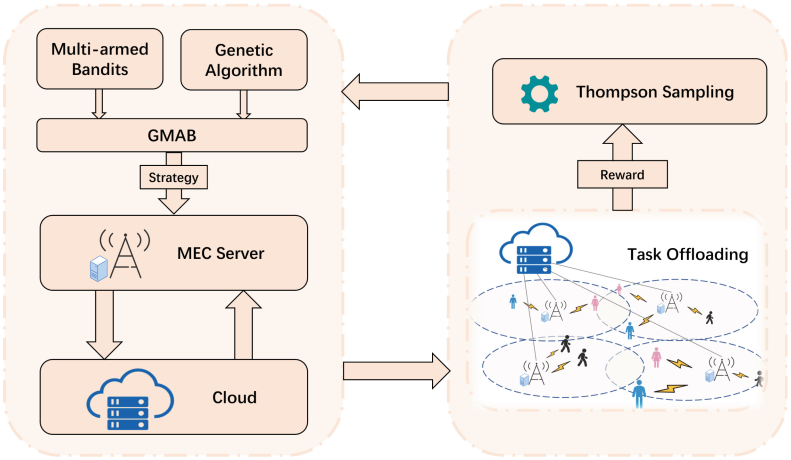

Fig. 2. Framework of GMAB for yielding the caching strategy.

As shown in Fig. 2, the bandit mechanism is deployed on each MEC server and it generates a new caching strategy for each time epoch to update the cache content from the cloud. The MEC environment generates a reward for the current strategy at each time epoch and provides feedbacks to Thompson Sampling, which ultimately determines the caching strategy for the next epoch.

4.1 Genetic Multi-armed Bandits Algorithm

At every epoch (with input T), Algorithm 1 determines if a caching update is necessary (Line 3). When an update is required, Q_{max} and R_{max} are checked for updates (Lines 4–7). The Bandit mechanism takes a new reward from $beta(\alpha, \beta)$ (Line 9) and searches for a new caching queue based on the reward through: 1) selecting the services with the largest reward to Q_2 (Lines 13–15); 2) shifting the focus to the uncached remaining services (Lines 17–20) for delayed opportunities of being selected. A genetic algorithm is used as well to cross Q_2 and Q_{max} for an improved convergence (Line 21). Finally, the MEC server caches the services that are placed in Q_2 and notifies its neighbors(Line 22). R is updated every epoch (Line 25). The modified genetic algorithm and the task offloading algorithm are discussed in Sect. 4.2 and 4.3, respectively.

Algorithm 1: Genetic Multi-armed Bandits Algorithm

```
   Input: time interval $T$, tasks of users $K$, MEC server $e_i$, application services $S$
1  Initialize beta distribution $b(\alpha, \beta)$, current optimal strategy $Q_{max}$, caching
   reward $R$
2  foreach episode do
3      if $t$ mod $T == 0$ then
4          if $R > R_{max}$ then
5              $Q_{max} \leftarrow Q_2$
6              $R_{max} \leftarrow R$
7          end
8          $R \leftarrow 0$
9          foreach $s \in S$ do
10             Sampling reward value $R_s$ from $b(\alpha, \beta)$
11         end
12         $Q_1 \leftarrow$ Sort services in descending order of $R$
13         $Q_2 \leftarrow \emptyset$, $c \leftarrow 0$
14         while $c < \frac{e_i[2]}{2}$ do
15             Select the service $s_c$ with the highest $R$ from $Q_1$ to place in Q2 and
               remove $s_c$ from Q1 $c \leftarrow c + 1$
16         end
17         $c \leftarrow 0$
18         while $c < \frac{e_i[2]}{2}$ do
19             Select the uncached service $s_c$ with the highest $R$ from $Q_1$ to place
               in Q2 and remove $s_c$ from Q1
20             $c \leftarrow c + 1$
21         end
22         Genetic_crossover()
23         Cache $Q_2$ into the server and synchronize information from adjacent
           servers
24     end
25     $R_q \leftarrow$ Environment_interaction()
26     $R \leftarrow R + R_q$
27 end
```

4.2 Modified Genetic Algorithm

As mentioned earlier, a modified genetic algorithm is incorporated for improving the convergence. As shown in Algorithm 2, it: 1) extracts services that are in Q_2 but not in Q_{max} (Lines 2–6); 2) extracts services that are in Q_{max} but not in Q_2 (Lines 7–11) and 3) puts them into S_1 and S_2.

Algorithm 2: Modified Genetic Algorithm

```
   Input: services list $Q_2$, pre-optimal strategy $Q_{max}$
1  Initialize $S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$
2  foreach $q \in Q_2$ do
3      if $q \notin Q_{max}$ then
4          Put $q$ into $S_1$
5      end
6  end
7  foreach $q \in Q_{max}$ do
8      if $q \notin Q_2$ then
9          Put $q$ into $S_2$
10     end
11 end
12 foreach $s \in S_2$ do
13     $a \leftarrow Random()$
14     if $a < 0.1$ then
15         Remove $S_1[0]$ form $Q_2$
16         Remove $S_1[0]$ form $S_1$
17         $Q_2 \leftarrow Q_2 \cup \{s\}$
18     end
19 end
20 return $Q_2$
```

4.3 Task Offloading Algorithm

We provided a detailed introduction of the task offloading process in Algorithm 3. When a task request arrives, the edge node e_i that is connected to the user, the neighbor N_{e_i} or the cloud can all respond and provide the required computing services. If e_i fails to hit the cache, this request is forwarded to N_{e_i} (Lines 2–7). When both e_i and N_{e_i} fail to hit or some errors occur on the edge side, the request is forwarded to the cloud (Lines 8-11). After all tasks are completed at the current time, the reward value R_s will be calculated by (10), and $b(\alpha, \beta)$ will be updated according to (11) (Lines 14–16). Finally, the reward value Rq of the caching strategy is calculated by (12) and returned to GMAB along with $b(\alpha, \beta)$(Lines 18-19).

$$R_i = \frac{n_{s_i}}{n_{max}} \cdot (\frac{k \cdot o_{s_i}}{\sum_{j=1}^{k} o_{s_j}} + \frac{k \cdot d_{s_i}}{\sum_{j=1}^{k} d_{s_j}}) \tag{10}$$

where n_{s_i} and n_{max} are the times of requesting service s_i and the max times in all services, respectively, o_{s_i} the average latency sensitivity of all tasks that requesting s_i and d_{s_i} the average up-link data size of all tasks that requesting s_i.

$$b(\alpha, \beta) = \begin{cases} b(\alpha+1, \beta), & (R_{ss} = 1) \\ b(\alpha, \beta+1), & (R_{ss} = 0) \end{cases} \tag{11}$$

Algorithm 3: Task Offloading Algorithm

```
   Input: tasks of users $K_t$, application services $S$
 1 foreach $k \in K_t$ do
 2     if the required services for $k$ are cached in server $e_i$ then
 3         forward $k$ to $e_i$
 4     end
 5     else if the required services for $k$ are cached in server $e_j \in N_{e_i}$ then
 6         forward $k$ to $e_j$
 7     end
 8     else
 9         forward $k$ to cloud
10     end
11     If the error occurs at the edge, the task will be forwarded to the cloud.
12 end
13 foreach $s \in S$ do
14     calculate reward $R_s$ for service $s$ according to (10)
15     map $R_s$ to $R_{ss}$ which can only be number 0 or 1
16     update parameters for $b(\alpha, \beta)$ according to (11)
17 end
18 calculate reward $R_q$ for current strategy according to (12)
19 return $R_q, b(\alpha, \beta)$
```

$$R_q = \sum_{j=1}^{k} (y_{i,j}^t \cdot R_i \cdot \frac{u_j}{b_i}) \tag{12}$$

where u_j is the size of s_j.

5 Performance Evaluation

We build a real-world simulation environment based on the Shanghai Telecom's base station data set, which contains 7.2 million internet access logs from 3,233 edge stations for 9,481 mobile users over 6 months. If a user has access to the internet, we can know when it sends a request and which base station the user is connected to. Figure 3 shows the distribution of nodes on the edge of Shanghai. We use Python to implement the proposed method. Chen *et al.* reported the round-trip time to the public cloud is 74 milliseconds, which we also used in our evaluation. We set the task's error rate at the edge to 1% and check for updates to caching every 5 epochs. To study the effect of the modified genetic algorithm, we design an experiment to compare the convergence speed of the algorithm, with and without the modified genetic algorithm. The parameters related to the simulation are shown in Table 2.

All the experiments are conducted on the same computer with an AMD Ryzen7 6800H 3.20 GHz processor, 16.0 GB of RAM, and using Python 3.10.

Table 2. Parameter table

Parameters	Value
Type of requests	800
Caching size of edge servers	0–450
Up-link data size(MB)	0.5–50
Channel bandwidth between edge servers(MHz)	20
Channel bandwidth between edge servers and cloud(MHz)	500
Signal transmission power(W)	0.5
Channel gain	$[D(e_i, r_j)]^4$
Computing amount of tasks(TFLOPs)	1.2-3.6
Computing capability of edge servers(TFLOPS)	0.4–1.2
Total rounds of simulation	500

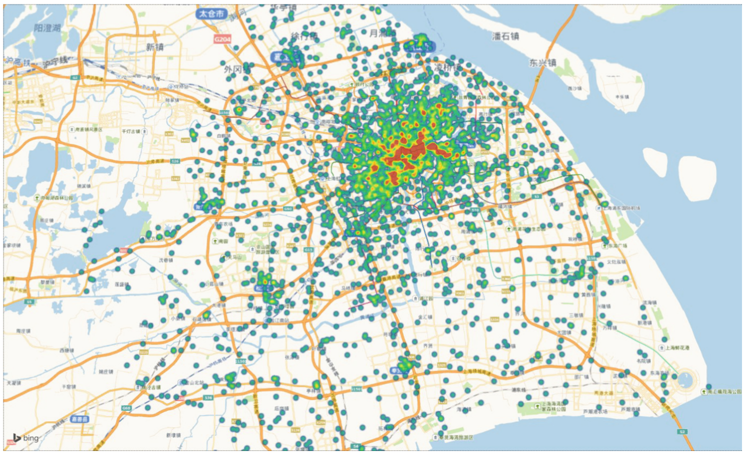

Fig. 3. Distribution of all 3,233 base stations in Shanghai. Each node denotes a base station.

We compared the performance of our proposed method with four baselines. We consider using the Oracle method as the first baseline, which possesses complete and accurate information about future service requests (i.e. has sufficient knowledge about future task types). This baseline is used to evaluate the effectiveness of our proposed methods in adapting to future changes in service requirements. The second baseline is the DCC-MAB algorithm proposed by Malazi *et al.* [12], which uses a modified UCB1 method to drive the bandit algorithm. The third method is based on the centralized multi-armed bandit method introduced by Chen *et al.* [14]. This method allows for the placement of each service in a fixed number of edge servers to meet budget constraints. The last baseline is the distributed collaborative service placement method presented in Yu *et al.* [21].

Figure 4 shows the caching hit rates of different strategies under different caching capacities of the MEC servers. It is obvious that the caching hit rate increases with increasing MEC server capacity. The Oracle method performs 2% higher than GMAB, while GMAB beats DCC-MAB/Chen *et al.*/Yu *et al.* by 5.2%/6.3%/
11.1%, respectively.

Figure 5 reveals that our proposed method has made significant improvement in response latency. The average response time of the Oracle method is 4% lower than GMAB, while GMAB produces an average response time that is 2% lower than that of the DCC-MAB method, and is 5.4%/15.6% lower than Chen *et al.*/Yu *et al.*, respectively.

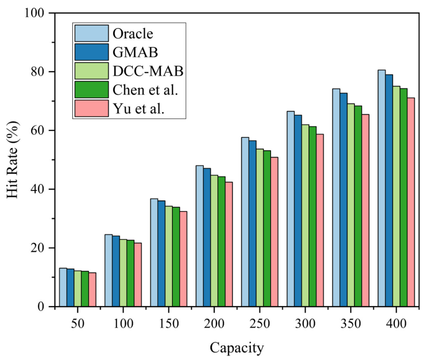

Fig. 4. Capacity and Hit Rate.

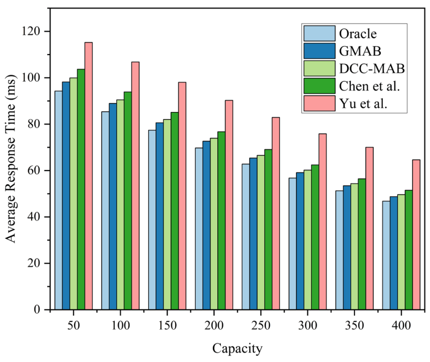

Fig. 5. Capacity and Response Time.

Figure 6 shows that the backhaul traffic of different algorithms shows a decreasing trend with the increase of caching capacity. The proposed GMAB approach has a backhaul traffic reduction of 7.1% compared to the baseline Oracle. The GMAB achieves the best performance in comparison with the other algorithms. Specifically, the backhaul traffic of GMAB averages 3.1% lower than DCC-MAB, 14.9% lower than that of Chen *et al.*, and 44.5% lower than that of Yu *et al.*.

Furthermore, we design an experiment to analyze the effectiveness of the modified genetic algorithm for MAB. The O-MAB refers to the GMAB method lacking the modified genetic algorithm. The plot in Fig. 7 illustrates that, despite the fact that both O-MAB and GMAB achieved the same hit rate eventually, the convergence speed of O-MAB was substantially lower than that of GMAB. Moreover, DCC-MAB performed even worse than O-MAB in terms of both convergence speed and hit rate.

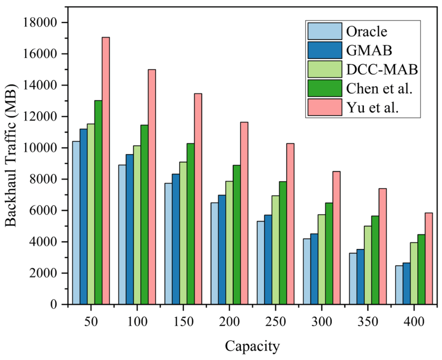

Fig. 6. Capacity and Backhaul Traffic.

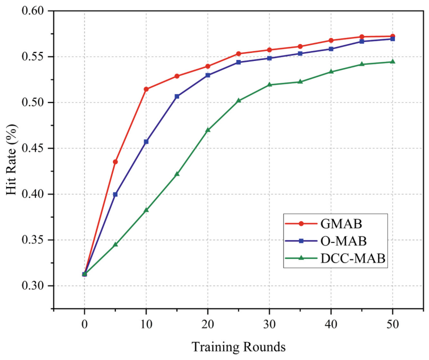

Fig. 7. Training Rounds and Hit Rate.

6 Conclusion

In this paper, we study the service caching problem in an MEC environment and present a Multi-Armed Bandits Learning-based caching method, GMAB. It comprises a genetic multi-armed bandits model for yielding high-quality caching schedules and a genetic algorithms for optimizing the convergence speed. The experimental results obtained upon a real-world date set of Shanghai Telecom's base station demonstrate that GMAB outperforms its peers in terms of multiple performance aspects.

References

1. Xu, X., Chen, P., Xia, Y., Long, M., Peng, Q., Long, T.: Mroco: a novel approach to structured application scheduling with a hybrid vehicular cloud-edge environment, in. IEEE Int. Conf. Serv. Comput. (SCC) **2022**, 84–92 (2022)
2. Ioannou, A., Weber, S.: A survey of caching policies and forwarding mechanisms in information-centric networking. IEEE Commun. Surv. Tutorials **18**(4), 2847–2886 (2016)
3. Ahlehagh, H., Dey, S.: Video caching in radio access network: impact on delay and capacity, in. IEEE Wirel. Commun. Network. Conf. (WCNC) **2012**, 2276–2281 (2012)
4. Xia, X., Chen, F., He, Q., Grundy, J., Abdelrazek, M., Jin, H.: Online collaborative data caching in edge computing. IEEE Trans. Parallel Distrib. Syst. **32**(2), 281–294 (2021)
5. Zhao, J., Sun, X., Li, Q., Ma, X.: Edge caching and computation management for real-time internet of vehicles: an online and distributed approach. IEEE Trans. Intell. Transp. Syst. **22**(4), 2183–2197 (2021)
6. Zeng, Y., et al.: Smart caching based on user behavior for mobile edge computing. Inf. Sci. **503**, 444–468 (2019)

7. Sengupta, A., Amuru, S., Tandon, R., Buehrer, R.M., Clancy, T.C., Learning distributed caching strategies in small cell networks. In: 11th International Symposium on Wireless Communications Systems (ISWCS). IEEE 2014, pp. 917–921 (2014)
8. Zhong, C., Gursoy, M.C., Velipasalar, S.: Deep reinforcement learning-based edge caching in wireless networks. IEEE Trans. Cogn. Commun. Network. **6**(1), 48–61 (2020)
9. Song, J., Sheng, M., Quek, T.Q., Xu, C., Wang, X.: Learning-based content caching and sharing for wireless networks. IEEE Trans. Commun. **65**(10), 4309–4324 (2017)
10. Wu, P., Li, J., Shi, L., Ding, M., Cai, K., Yang, F.: Dynamic content update for wireless edge caching via deep reinforcement learning. IEEE Commun. Lett. **23**(10), 1773–1777 (2019)
11. Qiao, G., Leng, S., Maharjan, S., Zhang, Y., Ansari, N.: Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. IEEE Internet Things J. **7**(1), 247–257 (2020)
12. Malazi, H.T., Clarke, S.: Distributed service placement and workload orchestration in a multi-access edge computing environment. IEEE Int. Conf. Serv. Comput. (SCC) **2021**, 241–251 (2021)
13. Wu, B., Chen, T., Yang, K., Wang, X.: Edge-centric bandit learning for task-offloading allocations in multi-rat heterogeneous networks. IEEE Trans. Veh. Technol. **70**(4), 3702–3714 (2021)
14. Chen, L., Xu, J., Ren, S., Zhou, P.: Spatio-temporal edge service placement: a bandit learning approach. IEEE Trans. Wireless Commun. **17**(12), 8388–8401 (2018)
15. Xu, H., Chen, R., Xu, M., Jiang, M., Lu, X.: Device-to-device collaborative caching strategy based on incentive mechanism. IEEE/CIC Int. Conf. Commun. China (ICCC) **2021**, 612–617 (2021)
16. Jiang, W., Feng, G., Qin, S.: Optimal cooperative content caching and delivery policy for heterogeneous cellular networks. IEEE Trans. Mob. Comput. **16**(5), 1382–1393 (2017)
17. Ren, D., Gui, X., Lu, W., An, J., Dai, H., Liang, X.: Ghcc: grouping-based and hierarchical collaborative caching for mobile edge computing. In: 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt). IEEE, pp. 1–6 (2018)
18. Ren, D., et al.: Hierarchical resource distribution network based on mobile edge computing, in 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2017, pp. 57–64 (2017)
19. Lin, X., et al.: 5G new radio: unveiling the essentials of the next generation wireless access technology. IEEE Commun. Standards Mag. **3**(3), 30–37 (2019). https://doi.org/10.1109/MCOMSTD.001.1800036
20. Wu, L.Y., Zhang, X.S., Zhang, J.L.: Capacitated facility location problem with general setup cost. Comput. Oper. Res., vol. 33, pp. 1226–1241, 2006. https://doi.org/10.1016/j.cor.2004.09.012
21. Yu, N., Xie, Q., Wang, Q., Du, H., Huang, H., Jia, X.: Collaborative service placement for mobile edge computing applications. In: IEEE Global Communications Conference, GLOBECOM 2018, Abu Dhabi, United Arab Emirates, December 9–13, 2018. IEEE, 2018, pp. 1–6. https://doi.org/10.1109/GLOCOM.2018.8647338