Learning-to-Rank Based Strategy for Caching in Wireless Small Cell Networks

Chenxi Zhang[1,2], Pinyi Ren[1,2](✉), and Qinghe Du[1,2]

[1] Department of Information and Communications Engineering, Xi'an Jiaotong University, Xi'an, China
zcx110708@hotmail.com, {pyren,duqinghe}@mail.xjtu.edu.cn
[2] Shaanxi Smart Networks and Ubiquitous Access Research Center, Xi'an, China

Abstract. Caching in wireless network is an effective method to reduce the load of backhaul link. In this paper, we studied the problem of wireless small cell network caching when the content popularity is unknown. We consider the wireless small cell network caching problem as a ranking problem and propose a learning-to-rank based caching strategy. In this strategy, we use the historical request records to learn the rank of content popularity and decide what to cache. First, we use historical request records to cluster the small base stations (SBS) through the k-means algorithm. Then the loss function is set up in each cluster, the gradient descent algorithm is used to minimize the loss function. Finally we can get the ranking order of the content popularity for each SBS, and the files are cached to the SBS in sequence according to the order. From Simulation results we can see that our strategy can effectively learn the ranking of content popularity, and obtain higher cache hit rate compared to the reference strategies.

Keywords: Wireless networks · Caching · Learning-to-rank

1 Introduction

As smart phones and mobile network develop, the main business of wireless network is transferred from traditional communication services to mobile data services, this makes the mobile network face heavier and heavier load pressure. In 5G networks, deploying small base stations in hotspots to relieve the load pressure of wireless networks is a common method, but there are still lots of problems to be solved. Especially the limited capacity of backhaul links has attracted the attention of many scholars. One solution to handle this problem is to deploy caching devices in wireless network edge devices to cache popular

The research reported in this paper is supported by the Key Research and Development Program of Shaanxi Province under Grant No. 2017ZDXM-G-Y-012 and the Fundamental Research Funds for the Central Universities.

B. Li et al. (Eds.): IoTaaS 2018, LNICST 271, pp. 111–119, 2019.
https://doi.org/10.1007/978-3-030-14657-3_12

content, such as base stations or mobile terminals, so that users can get those content directly from the caching devices when they request. This method can not only reduce the backhaul link's load pressure, but also reduce the file transfer delay and improve the user satisfaction [1,2,10].

In the wireless network caching technology, how to determine the popular content is a very important issue. Most of the research uses content popularity to determine the popular content. However, In the actual system, content popularity is constantly changing and difficult to get the accurate value in advance. So how to determine the content of caching without knowing of content popularity has become the focus of research. In [3], the author consider the cache problem as a multi-armed bandit problem, and propose a learning based caching algorithm. Then author in [8] take the context information into consideration. Author in [12] use the collaborative filtering algorithm in mobile social network, and propose a learning based caching strategy. But most of these works are focus on predict the value of content popularity.

When content popularity is already known, many researches modeled the caching problem as a 0–1 integer programming problem, and many researchers use greedy algorithm to handle this problem, such as [5,6,9]. Besides, the most popular caching strategy [8] which always caching the files with highest content popularity. We can find that many caching strategies are caching the most popular file or the file have highest gain to the system. So in this paper, we model the wireless small cell network caching problem as a ranking problem, no longer focusing on the value of content popularity, but is the ranking relationship between the content popularity. And we solve this problem by a learning-to-rank method [11].

The main contribution of this paper are as follows. We model the caching problem in wireless small cell network as a sorting problem when the system have no knowledge of content popularity. And we give a learning-to-rank based algorithm to solve this problem. In the simulation results, we can find that our strategy can get higher cache hit rate than reference strategies and close to the optimal strategy.

Rest of this paper is organized as follows: In Sect. 2 we give the system model and the optimization problem. Section 3 we describe the details of the learning-to-rank based caching strategy. Section 4 we simulate the propose strategy and compared with some existing strategies. Finally, we make a summary of this paper.

2 System Model

We consider a wireless small cell network with M SBSs which are distributed in one area, assuming that the coverage of these SBSs does not overlap, and each one of them is equipped with a caching device, which can cache popular files and connect to the core network through the backhaul link. When a user in a SBS's service range requests a file, the system first look for the file on the SBS's cache device, if found, it will be sent directly to the user by the SBS. If the file is not cached, the SBS will request the file from the core network, then the core network will send the file to SBS through the backhaul link and send it to the user.

All users request files from the file library $F = \{f_1, f_2 \ldots, f_N\}$ with a total number of N, each file have a size of $S_f = \{s_1, s_2 \ldots, s_N\}$. All the SBS equipped a cache device with cache memory of S Mb. We assume that the small base stations in the system are distributed in different areas, such as schools, factories, office buildings, etc. Therefore, users of different SBS may have different content preferences, so the content popularity in different SBS is not the same. The probability that the user is in the service range of each SBS is equal. We use $P \in \mathbb{R}^{M \times N}$ to represent the content popularity of all SBSs, which the element $p_{m,i}$ represents the probability of the user in i-th SBS's coverage request for i-th file. And we use the matrix X to represent the cache matrix, the element $x_{m,i} = 1$ represents the m-th SBS cached the i-th file, otherwise is 0.

We aim to maximize the cache hit rate in our system, which is the probability of finding the corresponding file on the SBS's cache device when the user requests a file. So the optimization problem for our system is as follows:

$$\begin{aligned} \max_{X} \quad & CHR = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{N} p_{m,i} x_{m,i} \\ s.t. \quad & \sum_{i=1}^{N} s_i x_{m,i} \leq S, m = 1, 2, \cdots, M, i = 1, 2, \cdots, N \\ & x_{m,i} \in \{0, 1\}, m = 1, 2, \cdots, M, i = 1, 2, \cdots, N \end{aligned} \tag{1}$$

We can see that the optimization variable of this optimization problem is a binary variable. With known of the content popularity P, the optimization problem is a 0–1 integer programming problem. The 0–1 integer programming problem is a typical NP problem [7], there are many algorithm can solve this problem, such as greedy algorithm, branch and bound. But in our system, content popularity P is not a known parameter, so solving this problem becomes very difficult. In next section we propose a strategy based on a learning-to-rank to solve this problem.

3 Learning-to-Rank Based Caching Strategy

In this section, we propose a cache strategy based on learning-to-rank. Learning-to-rank is a kind of machine learning method that used for information retrieval [4]. We use the historical request record to learn the rank of content popularity and decide what to cache.

The caching strategy proposed in this section has two steps. The first step is to cluster the SBSs, which replaces the process of feature extraction in learning-to-rank [4]. Next step is to determine the caching files of each SBS based on the historical request record in each cluster by learning-to-rank method.

3.1 k-means Based Small Base Station Clustering

We cluster the SBSs according to the historical request record $R \in \mathbb{R}^{M \times N}$, $r_{i,j}$ represents the number of requests to j-th file received by i-th SBS, and R_i represent the i-th row of the historical request record which is the historical request record of the i-th SBS. We normalize the historical request record matrix R, so as to improve the clustering accuracy, and get a new matrix R^*,. Then we use k-means algorithm to cluster the SBSs, the process is as follows:

(1) We initialize k clusters $C_1, C_2 \ldots C_k$, and randomly select k rows from R^* as the initial centroid $\mu_1^{(1)}, \mu_2^{(2)} \ldots \mu_k^{(k)}$ of these clusters.
(2) We allocate each SBS into the cluster which have the smallest Euclidean distance between the SBS's historical request record and the centroid.

$$C_m = \{R_i^* : \|R_i^* - \mu_m^{(t)}\|_2 \leq \|R_i^* - \mu_n^{(t)}\|_2, 1 \leq n \leq k\} \tag{2}$$

(3) According to the clustering results, we can get the new centroid of each cluster which is calculated by the follow formula:

$$\mu_m^{(t+1)} = \frac{1}{|C_m|} \sum_{R_i^* \in C_m} R_i^* \tag{3}$$

Repeat the step 2 and step 3, until the centroid of all the clusters is unchanged. Then we can get the cluster results and each cluster's historical request record $H_1, H_2 \ldots, H_k$. And the SBSs in the same cluster with similar content popularity.

3.2 Learning-to-Rank Based Caching Algorithm

In this step, we use the learning-to-rank method to enable the base station to learn the ranking of content popularity and decide what file to cache for each cluster.

We assume there is a final ranking matrix $H_m^* = U^T V$ for each cluster. And we use the top one probability [4] to calculate the probability of the highest number of requests for each file, the formula is as follows:

$$P_{H_m(i)}(h_{i,j}^m) = \frac{\varphi(h_{i,j}^m)}{\sum_{n=1}^{N} \varphi(h_{i,n}^m)} \tag{4}$$

The $\varphi(x)$ represents a monotone increasing and constant positive function, in this paper we define this function as an exponential function.

Then the cross entropy loss function in our system can be obtained as follows:

$$\begin{aligned} L(U,V) &= \sum_{i=1}^{M_m}\{-\sum_{j=1}^{N} P_{H_m(i)}(h_{i,j}^m)\log P_{H_m(i)}(g(U_i^T V_j))\} + \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) \\ &= \sum_{i=1}^{M_m}\{-\sum_{j=1}^{N} I_{ij}\frac{exp(h_{i,j}^m)}{\sum_{n=1}^{N} I_{in}exp(h_{i,n}^m)}\log\frac{exp(g(U_i^T V_j))}{\sum_{n=1}^{N} I_{in}exp(g(U_i^T V_n))}\} \\ &+ \frac{\lambda}{2}(\|U\|_F^2 + \|V\|_F^2) \end{aligned} \tag{5}$$

where $g(x)$ represents the sigmoid function, which is $g(x) = 1/(1+e^{-x})$, and λ represents the regularization coefficient. The value of the assumed probability distribution H_m^* can be obtained by minimizing this loss function. In order to We use the gradient descent algorithm to minimize the loss function, the gradient is shown in the following form:

$$\begin{aligned} &\frac{\partial L(U,V)}{\partial U_i} = \\ &\sum_{j=1}^{N} I_{ij}(\frac{exp(g(U_i^T V_j))}{\sum_{n=1}^{N} I_{in}exp(g(U_i^T V_n))} - \frac{exp(h_{i,j}^m)}{\sum_{n=1}^{N} I_{in}exp(h_{i,n}^m)})g^{'}(U_i^T V_j)V_j + \lambda U_i \\ &\frac{\partial L(U,V)}{\partial V_j} = \\ &\sum_{i=1}^{M_m} I_{ij}(\frac{exp(g(U_i^T V_j))}{\sum_{n=1}^{N} I_{in}exp(g(U_i^T V_n))} - \frac{exp(h_{i,j}^m)}{\sum_{n=1}^{N} I_{in}exp(h_{i,n}^m)})g^{'}(U_i^T V_j)U_i + \lambda V_j \end{aligned} \tag{6}$$

We can update the value of U and V until the loss function is convergence, and the updating formula are as follows:

$$\begin{aligned} U_i &= U_i - \alpha\frac{\partial L(U,V)}{\partial U_i} \\ V_j &= V_j - \alpha\frac{\partial L(U,V)}{\partial V_j} \end{aligned} \tag{7}$$

Finally, we can get the value of U and V, then we can calculate $P_i = U_i^T V$ to get the ranking of the files for each SBS by sort the value of each element of P_i in descending order. According the ranking of the files for each SBS, we can cache the files in descending unit the cache memory is filled. The whole process of our algorithm are given in Algorithm 1.

Algorithm 1. Learning-to-Rank Based Caching Algorithm

```
1: **Inputs:** historical request record $R$,cluster number $k$
2: Cluster the SBSs by k-means algorithm
3: Get historical request record $H_1, H_2 \ldots, H_k$
4: for $m = 1, 2 \cdots, k$ do
5:    initialize the loss function by (5)
6:    while Loss function is not convergence do
7:       Calculate the gradient by (6)
8:       Update $U$ and $V$ by (7)
9:    end while
10:   for $i = 1, 2 \cdots, M_m$ do
11:      Get the ranking of the files by (8)
12:      while $S_M < S$ do
13:         Cache file to the SBS
14:      end while
15:   end for
16: end for
17: Output: cache matrix $X$
```

4 Simulation Results

In this section, we simulate the performance of the proposed strategy and compared it with some existing algorithms.

We assume that there are 100 SBSs which are divided into four categories, each with a different Zipf distribution file order. Each SBS has different Zipf distribution parameters, which follow a uniform distribution with a mean of 0.5. The learning rate of gradient descent algorithm is 0.5, and the regularization coefficient $\lambda = 0.8$. There is 100 files in the file library, and all files with size of 1 Mb. The capacity of the cache device in each SBS is 20 Mb. And each SBS receive 200 times request from the user to set up the historical request record.

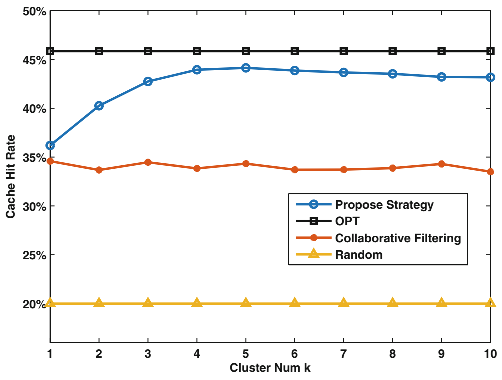

Fig. 1. Cache hit rate varies with cluster number k.

To better observe the simulation results of the proposed strategy, we compared our strategy with three existing strategies. First is the OPT strategy, which let the SBSs known the content popularity in advance and cache the optimal files. Next is the random cache strategy, in this way the SBSs will cache some files by random. Finally is the collaborative filtering based cache strategy [12], this strategy uses collaborative filtering algorithm to decide which file should cache. The idea of the collaborative filtering algorithm is the SBSs have similar request records will request the same files in the future. All simulation results are averaged after repeating 1000 times.

In Fig. 1 we simulate the cache hit rate in different number of cluster. From the Fig. 1, we can see that when the number of clusters k = 1, the SBSs is not clustered, the cache hit rate of the system is 36% when our strategy is used, slightly higher than the collaborative filtering strategy. With increase of the cluster number, the cache hit rate of propose strategy is gradually rising, and it reaches the maximum of about 44.5% when the number of clusters is between 4 and 5. This is also in accordance with our simulation settings. When the cluster number is higher than 5, the cache hit rate has decreased, this is because the number of SBSs in each cluster is decreased. The cluster numbers can be decided by the elbow method.

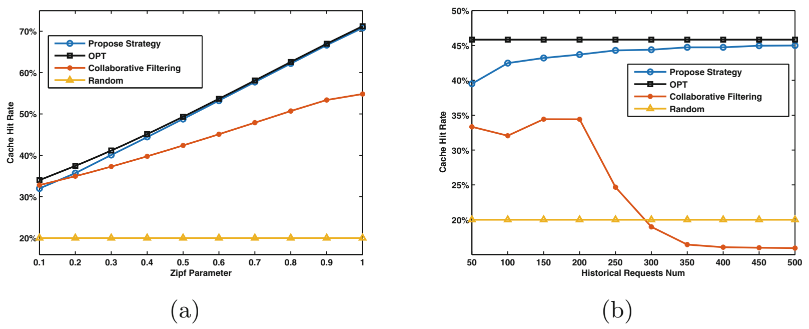

Fig. 2. (a) Cache hit rate varies with Zipf parameter (b) Cache hit rate varies with historical requests numbers

Next, we analyze the influence of Zipf distribution parameters on the system performance. From Fig. 2(a) we can see that, when the Zipf parameter is less than 0.2, the cache hit rate in propose strategy is slightly less than the strategy based on collaborative filtering. This is because when the Zipf parameter is very small the request probability of each file is almost same, so it is difficult to learn the rank of content popularity. With the increase of Zipf parameters, the cache hit rate obtained in propose strategy is more than the strategy based on collaborative filtering and close to the optimal strategy. Finally, we present the simulation results of the cache hit rate varies with the historical request numbers in Fig. 2(b). In this picture, it can be seen that when the number of requests is only 50 times, the proposed strategy can get a cache hit rate of close to 40%. As the number of requests increase, the cache hit rate of the proposed strategy is also increasing, and the cache hit rate is about 45% when the number of requests is 500 times. On the other hand, the performance of strategy based on collaborative filtering is much lower than random cache strategy, when the numbers of requests is higher than 300 times, this is because the collaborative filtering strategy only considers whether the file was requested rather than the numbers of requests. So we can get the conclusion that the strategy proposed in our paper has stronger learning ability and can make the system performance close to the performance of the best strategy by learning.

5 Conclusions

In this paper, we propose a learning-to-rank based caching strategy for wireless small cell networks. This strategy consider the wireless small cell network caching problem as a sorting problem, and the history request record is used to learn the rank of content popularity. Then the files are cached to the small base station in sequence according to the order. Simulation shows that this strategy can effectively learn content popularity ranking, thus caching the appropriate content, and the performance is close to the optimal cache strategy. Compared to the reference caching strategy this strategy can adapt to more scenes and obtain higher cache hit rate.

References

1. Bastug, E., Bennis, M., Debbah, M.: Living on the edge: the role of proactive caching in 5G wireless networks. IEEE Commun. Mag. **52**(8), 82–89 (2014)
2. Bastug, E., et al.: Big data meets telcos: a proactive caching perspective. J. Commun. Netw. **17**(6), 549–557 (2016)
3. Blasco, P., Gunduz, D.: Learning-based optimization of cache content in a small cell base station. In: IEEE International Conference on Communications, pp. 1897–1903 (2014)
4. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: International Conference on Machine Learning, pp. 129–136 (2007)
5. Jiang, D., Cui, Y.: Partition-based caching in large-scale SIC-enabled wireless networks. IEEE Trans. Wirel. Commun. **PP**(99), 1 (2017)
6. Keshavarzian, I., Zeinalpour-Yazdi, Z., Tadaion, A.: A clustered caching placement in heterogeneous small cell networks with user mobility. In: IEEE International Symposium on Signal Processing and Information Technology, pp. 421–426 (2016)
7. Korbut, A.A., Sigal, I.K.: Exact and greedy solutions of the knapsack problem: the ratio of values of objective functions. J. Comput. Syst. Sci. Int. **49**(5), 757–764 (2010)
8. Muller, S., Atan, O., Schaar, M.V.D., Klein, A.: Context-aware proactive content caching with service differentiation in wireless networks. IEEE Trans. Wirel. Commun. **16**(2), 1024–1036 (2017)

9. Sermpezis, P., Spyropoulos, T., Vigneri, L., Giannakas, T.: Femto-caching with soft cache hits: improving performance with related content recommendation. In: 2017 IEEE Global Communications Conference, pp. 1–7 (2018)
10. Shanmugam, K., Golrezaei, N., Dimakis, A.G., Molisch, A.F., Caire, G.: Femto-caching: wireless content delivery through distributed caching helpers. IEEE Trans. Inf. Theory **59**(12), 8402–8413 (2011)
11. Shi, Y., Larson, M., Hanjalic, A.: List-wise learning to rank with matrix factorization for collaborative filtering. In: ACM Conference on Recommender Systems, Recsys 2010, September, Barcelona, Spain, pp. 269–272 (2010)
12. Wang, Y., Ding, M., Chen, Z., Luo, L.: Caching placement with recommendation systems for cache-enabled mobile social networks. IEEE Commun. Lett. **PP**(99), 1 (2017)