Smart Caching in Wireless Small Cell Networks via Contextual Multi-Armed Bandits

Sabrina Müller*, Onur Atan†, Mihaela van der Schaar†, Anja Klein*
*Communications Engineering Lab, TU Darmstadt, Germany, {s.mueller, a.klein}@nt.tu-darmstadt.de
†Department of Electrical Engineering, University of California Los Angeles, USA,
oatan@ucla.edu, mihaela@ee.ucla.edu

***Abstract*—A promising architecture for content caching in wireless small cell networks is storing popular files at small base stations (sBSs) with limited storage capacities. Using localized communication, an sBS serves local user requests, while reducing the load on the macro cellular network. The sBS should cache the most popular files to maximize the number of cache hits. Content popularity is described by a popularity profile containing the expected demand of each file. Assuming a fixed popularity profile of which the sBS has complete knowledge, the optimal content placement problem reduces to ranking the files according to their expected demands and caching the highest ranked ones. Instead, we assume that the popularity profile is varying, for example depending on fluctuating types of users in the vicinity of the sBS, and it is unknown a priori. We present a novel algorithm based on contextual multi-armed bandits, in which the sBS regularly updates its cache content and observes the demands for cached files in different contexts, thereby learning context-dependent popularity profiles over time. We derive a sub-linear regret bound, proving that our algorithm learns smart caching. Our numerical results confirm that by exploiting contextual information, our algorithm outperforms reference algorithms in various scenarios.**

I. Introduction

Global mobile data traffic has been increasing over the past years and is predicted to grow almost tenfold within the next few years [1]. One of the major generators of mobile data traffic is video traffic. While mobile video traffic accounts for more than half of the mobile data traffic already today, it is predicted to constitute almost three quarters of mobile data traffic in a few years [1]. A solution to alleviate highly-loaded networks from the burdens of mobile data traffic is *caching at the edge* [2]. The most popular files, such as popular videos, are stored at local caches to serve end users' requests directly via localized communication. By bringing content closer to the end user, bandwidth requirements and delay can be reduced. Recent work showed that a promising caching architecture is given by small cell networks in which *small base stations* (sBSs) are utilized as local caching entities [3], [4]. These sBSs dispose of limited storage capacities. In their caches, they can store a fraction of the available content to serve users in their vicinity via localized communication. In this way, the load on the macro cellular network is reduced.

Due to the vast amount of content available in wireless multimedia applications such as YouTube and Netflix, not all available files can be stored in a local cache. Hence, a crucial question for content caching is which files to store. A first line of literature investigates the problem of cache content placement in various caching scenarios under the assumption that the popularity profile is fixed and known in advance, i.e., the set of expected demands of available files is given, or even the actual future demands are known. In this way, the problem of cache content placement can be formulated as an optimization problem. In [2], distributed approximation algorithms for cache content placement in a hierarchical tree cache network are presented. The goal is to minimize the total bandwidth cost in the network. For this purpose, each leaf-cache in the tree can either store a file or fetch it upon request from another cache or from the main cache. In [3], a small cell network with several sBSs serving as caches is studied. Exploiting that users can be connected to several sBSs at the same time, content should be distributed among the caches such that the delay experienced by the users is minimized. An approximation algorithm for this problem is given. Building upon the same caching architecture, in [4], a distributed algorithm for cache content placement is proposed when users move in between sBSs. Since the users cannot receive the whole content from one single sBS, encoded parts of the content are stored in the sBSs. An approximate algorithm is presented that minimizes the probability that users have to request parts of the file from the main base station instead of the sBSs.

Assuming the popularity profile of files is given in advance requires knowledge about the expected demand of each file. Therefore, a second line of literature deals with the more realistic case of cache content placement without prior knowledge about file popularity, but still assuming a fixed popularity profile. Applying methods from machine learning, algorithms are developed that learn the optimal cache content placement over time. In [5], using a multi-armed bandit algorithm, an sBS learns the fixed popularity profile online by refreshing its cache content and observing instantaneous demands for cached files over time. In this way, the cache content placement at an sBS is optimized over time to maximize the traffic that can be served by the sBS. The authors extend their framework for a similar scenario in [6] and [7], where they additionally take into account the costs for adding new files to the cache. Moreover, they provide theoretical sub-linear regret bounds for their algorithms. In references [5] - [7], the optimal caching strategy is learned over time based on previous observations of instantaneous demands. On the contrary, social correlations are exploited in [8], [9] by taking into account user ratings as available for example in social networks. However, they only consider one period in time in which the popularity profile is estimated based on a training set of ratings instead of learning online by updating the popularity estimates over time. Moreover, all aforementioned approaches do not take into account the potential diversity of file popularity among different user types or at different points in time. Popularity diversity among user types is taken into account in [10], where users are clustered into groups of similar interests. Each user group is assigned to an sBS which then learns the popularity profile of its user group. However, the popularity profile which is learned by each sBS is again fixed.

In this paper, we propose a novel online learning algorithm for cache content placement based on a *contextual multi-armed bandit problem.* Algorithms for different variants of the contextual multi-armed bandit problem have been developed before, e.g. for click-through rate maximization in web search [11], [12] or distributed online learning [13]. Our proposed algorithm for smart caching at an sBS is inspired by the distributed contextual learning algorithm presented in [13], which considers a general multi-agent learning setting, while we focus on a single learner. In [13], a learner selects one action at a time, while we extend the case to taking multiple actions at a time, since the sBS has to select multiple files to cache. In our proposed algorithm, an sBS learns the best caching strategy to maximize the average number of cache hits by observing *context-dependent* instantaneous demands *over time*. By taking into account *context information*, the sBS learns the specific file popularity of different contexts, instead of averaging popularity estimates over all possible contexts. The context information considered here can include global information about the system state (e.g., the time of the day or the time in the week, the number of users in the vicinity of the sBS), but also characteristics related to each single file (e.g., how often it has been shared in social media), or characteristics of users in the vicinity of the sBS (e.g., their ages, their genders, their interests, their ratings).

The contributions of this paper are as follows:

- We present a novel caching algorithm based on contextual multi-armed bandit optimization to maximize the number of cache hits at an sBS.
- We derive a sub-linear regret bound for the caching algorithm, which proves that our algorithm learns smart caching.
- We numerically evaluate the caching algorithm in different settings. A comparison shows that by exploiting contextual information, our algorithm outperforms reference algorithms.

The remainder of the paper is organized as follows. In Section II we describe the system model and give the problem formulation for smart caching at an sBS. In Section III we propose an online learning algorithm based on contextual multi-armed bandit optimization that exploits context information to learn the optimal caching strategy over time. In Section IV we derive a sub-linear bound on the regret of the learning algorithm. Numerical results of the learning algorithm are presented in Section V. Section VI concludes the paper.

II. System Model

Consider an sBS with a cache memory in a wireless network. The sBS has a reliable backhaul link to the core network. In its cache memory, the sBS can store up to m files from a set $F = \{1, ..., |F|\}$, where we assume that all files are of the same size. To inform users in its vicinity about available files, the sBS broadcasts the information about currently cached files periodically [5]. When a user requests a file that the sBS stored in its cache, the sBS serves the user via localized communication. In this case, no additional load is put on the macro cellular network. Otherwise, the user directly downloads the file from the macro cellular network. In order to relieve load from the macro cellular network, the sBS aims at optimizing the cache content such that the traffic it can serve directly is maximized. Maximizing the traffic served by the sBS corresponds to maximizing the number of *cache hits*, i.e., the number of requests for files cached at the sBS. Note that the sBS can only observe the requests for cached files, i.e., cache hits, but it cannot observe the requests for non-cached files, i.e., *cache misses*. For this purpose, over time the sBS should learn which files are most popular in which context, by observing not only demands for cached files, but also in which context they occur.

We consider a system that works in a discrete time setting $t = 1, 2, ..., T$, with finite time horizon T, where the following events happen sequentially, in each period t: (i) The sBS observes a D-dimensional context vector $x_t \in \mathcal{X}$ which characterizes the system in this period, where $\mathcal{X}$ is a bounded D-dimensional context space. Possible context dimensions are for example the time of the day or the time in the week, the number of users in the vicinity of the sBS, the users' ages, their genders or their ratings. (ii) Based on the context vector $x_t \in \mathcal{X}$, the sBS refreshes the cache content. The sBS then sends a broadcast message to all users in the vicinity informing about the set of currently cached files, which is denoted by $C_t = \{c_{1,t}, ..., c_{m,t}\}$. (iii) The sBS observes the demands $d_{c_{i,t}}(x_t, t)$ for all files $c_{i,t} \in C_t$ in this period, i.e., the number of requests for each single file currently stored in the cache. Then it provides the users with their requested files according to their demands.

Since the context space $\mathcal{X}$ is assumed to be bounded, it can be set to $\mathcal{X} := [0, 1]^D$ without loss of generality. When file $f \in F$ is stored in the cache after observing a context $x \in X$, the sBS observes the random demand $d_f(x)$, which is sampled from an unknown distribution depending on the context x. The demand is assumed to take values in $[0, U_{\max}]$, where $U_{\max}$ is the maximum number of users that can be served by the sBS. The expected demand for file f given context x is denoted by $\mu_f(x)$. In time slot t, the random variable $d_f(x_t)$ is assumed to be independent of all past caching decisions and previous demands. Since the sBS aims at maximizing the total number of cache hits, the demands for cached files can be seen as rewards which the sBS receives for cache hits. The goal of the sBS is to optimize the cache content to maximize the total number of cache hits up to the finite time horizon T. Suppose that for each context, the expected demands of all files would be known. Then, for each context, the optimal solution would be to cache the m files with highest expected demands, which is formalized as follows. For context $x \in \mathcal{X}$, we define the *top-m files for context x* as the following m files $f_1^*(x), f_2^*(x), ..., f_m^*(x) \in F$ which satisfy [1]

$$\begin{aligned} f_1^*(x) &\in \operatorname*{argmax}_{f \in F} \mu_f(x) \\ f_2^*(x) &\in \operatorname*{argmax}_{f \in F \setminus \{f_1^*(x)\}} \mu_f(x) \\ &\vdots \\ f_m^*(x) &\in \operatorname*{argmax}_{f \in F \setminus \{f_1^*(x), ..., f_{m-1}^*(x)\}} \mu_f(x). \end{aligned} \tag{1}$$

Then, an optimal choice of files to cache given context x is defined by the files in (1). However, we assume that the expected demands are unknown a priori. In this case, the sBS has to learn the expected demands over time. The optimal solution given in (1) can then serve as a benchmark to evaluate the loss of learning the expected demands instead of knowing them a priori. Below, this loss will be defined as the regret of learning. Under the assumption that the context-dependent expected demands are unknown a priori, the sBS has to learn these expected demands over time. For this purpose, the sBS has to find a trade-off between caching files about which few information is available (*exploration*) and files of which it believes that they will yield the highest demands (*exploitation*). In each time period, the choice of files to be cached depends on the history of choices in the past and the corresponding observed demands. An algorithm which maps the history to the choices of files to cache is called a *learning algorithm.*

Recall that $C_t = \{c_{1,t}, ..., c_{m,t}\}$ is the set of cached files chosen in time period t according to the learning algorithm of the sBS. The regret of learning with respect to the optimal benchmark solution is given by

$$R(T) = \sum_{t=1}^{T} \left(\sum_{i=1}^{m} \mu_{f_i^*(x_t)}(x_t) \right) - E\left(\sum_{t=1}^{T} \sum_{i=1}^{m} d_{c_{i,t}}(x_t, t) \right), \tag{2}$$

where $d_{c_{i,t}}(x_t, t)$ denotes the random demand for the cached file $c_{i,t} \in C_t$ for context x_t at time t. Here, the expectation is taken with respect to the choices made by the learning algorithm of the sBS and the distributions of the demands.

III. A Uniform Context Partitioning Algorithm for Smart Caching

The basic idea of the algorithm for smart caching at an sBS is as follows: The algorithm partitions the context space uniformly into smaller sets. Then, the sBS learns the expected demands for files independently in each of the sets, by estimating the expected demands for files based on the observed demands when context arrived from that set. In the algorithm, a time period t can either be an exploration or an exploitation phase. In exploration phases, the sBS chooses a random set of files to cache. Theses phases are needed to update the estimated demands also for files with low estimated demands. In exploitation phases, the sBS caches the files with the highest estimated demands. The algorithm for selecting m files is called *m- Contextual Learning with Uniform Partition* (m- CLUP) and its pseudocode is given in Figure 1.

[1]Several files may have the same expected demands, i.e., the optimal set of files may not be unique. This is also captured here.

m-CLUP for Smart Caching

```
1: Input: $T$, $s_T$, $K(t)$
2: Initialize context partition: Create partition $\mathcal{P}_T$ of context space $[0,1]^D$ into $(s_T)^D$ hypercubes of identical size
3: Initialize counters: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $N_{f,p} = 0$
4: Initialize estimates: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $\hat{d}_{f,p} = 0$
5: for each $t = 1, ..., T$ do
6:     Observe context $x_t$
7:     Find the set $p \in \mathcal{P}_T$ such that $x_t \in p$
8:     Compute the set of under-explored files $F_p^{ue}(t)$ in (3)
9:     if $F_p^{ue}(t) \neq \emptyset$ then
10:        $u = \text{size}(F_p^{ue}(t))$
11:        if $u \geq m$ then
12:            Select $c_{1,t}, ..., c_{m,t}$ randomly from $F_p^{ue}(t)$
13:        else
14:            Select $c_{1,t}, ..., c_{u,t}$ as the $u$ files from $F_p^{ue}(t)$
15:            Select $c_{u+1,t}, ..., c_{m,t}$ as the $(m-u)$ files $\hat{f}_{1,p}(t), ..., \hat{f}_{m-u,p}(t)$ from (4)
16:        end if
17:    else
18:        Select $c_{1,t}, ..., c_{m,t}$ as the $m$ files $\hat{f}_{1,p}(t), ..., \hat{f}_{m,p}(t)$ from (5)
19:    end if
20:    Observe user demands $d_1, ..., d_m$ for files $c_{1,t}, ..., c_{m,t}$
21:    for i=1,...,m do
22:        $\hat{d}_{c_{i,t},p} = \frac{\hat{d}_{c_{i,t},p} + d_i}{N_{c_{i,t},p} + 1}$
23:        $N_{c_{i,t},p} = N_{c_{i,t},p} + 1$
24:    end for
25: end for
```

Fig. 1. Pseudocode for m-CLUP

Next, we describe the algorithm in more detail. In its initialization phase, m-CLUP creates a partition $\mathcal{P}_T$ of the context space $[0,1]^D$ into $(s_T)^D$ D-dimensional hypercubes of identical size $\frac{1}{s_T} \times \ldots \times \frac{1}{s_T}$. Here, s_T is an input parameter which determines the number of sets in the partition. Additionally, m-CLUP keeps a counter $N_{f,p}(t)$ for each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$. The counter $N_{f,p}(t)$ is the number of periods in which file $f \in F$ was cached after a context from set p arrived up to period t. Moreover, m-CLUP initializes the estimated demand $\hat{d}_{f,p}(t)$ of each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$ up to period t. This estimated demand is calculated as follows: Let $\mathcal{E}_{f,p}(t)$ be the set of demands observed when file f was cached after a context from set p arrived up to period t. Then, the estimated demand of file f in set p is given by the sample mean $\hat{d}_{f,p}(t) := \frac{1}{|\mathcal{E}_{f,p}(t)|} \sum_{d \in \mathcal{E}_{f,p}(t)} d$.[2]

In each time period t, m-CLUP determines the set $p(t) \in \mathcal{P}_T$, from which the context arrived in this period, i.e., such that $x_t \in p(t)$ holds. Then, the algorithm can be in one of the two phases mentioned above, in an exploration phase or in an exploitation phase. In order to determine the correct phase for the current period, the algorithm checks if there are files that have not been explored sufficiently often. For this purpose, the *set of under-explored files* $F_p^{ue}(t)$ is calculated based on

$$F_p^{ue}(t) := \{f \in F : N_{f,p}(t) \leq K(t)\}, \tag{3}$$

where $K(t)$ is a deterministic, monotonically increasing control function, which is an input to the algorithm. The control function has to be set correctly to balance the trade-off between exploration and exploitation. In Section IV, we will select a control function that guarantees a good balance in terms of this trade-off. If the set of under-explored files is non-empty, m-CLUP enters the exploration phase. Let $u(t)$ be the size of the set of under-explored files. If the set of under-explored files contains at least m elements, i.e., $u(t) \geq m$, the algorithm randomly selects m files from $F_p^{ue}(t)$ to cache. If the set of under-explored files contains less than m elements, $u(t) < m$, it selects all $u(t)$ files from $F_p^{ue}(t)$ to cache. Since the cache is not fully filled by $u(t) < m$ files, additionally $(m - u(t))$ other files can be cached. In order to exploit knowledge obtained so far, m-CLUP selects $(m - u(t))$ files from $F \setminus F_p^{ue}(t)$ with highest estimated demands, as defined by the files $\hat{f}_{1,p}(t), ..., \hat{f}_{m-u(t),p}(t) \in F \setminus F_p^{ue}(t)$, for which the following holds for $j = 1, ..., m - u(t)$:

$$\hat{f}_{j,p}(t) \in \underset{f \in F \setminus (F_p^{ue}(t) \cup \bigcup_{i=1}^{j-1} \{\hat{f}_{i,p}(t)\})}{\operatorname{argmax}} \hat{d}_{f,p}(t), \tag{4}$$

where $\bigcup_{i=1}^{0} \{\hat{f}_{i,p}(t)\} = \emptyset$. If the set of files defined by (4) is not unique, ties are broken arbitrarily. Note that by this procedure, even in exploration phases, the algorithm additionally exploits, whenever the number of under-explored files is smaller than the cache size. If the set of under-explored files $F_p^{ue}(t)$ is empty, m-CLUP enters the exploitation phase. It selects m files from F with highest estimated demands as defined by the files $\hat{f}_{1,p}(t), ..., \hat{f}_{m,p}(t) \in F \setminus F_p^{ue}(t)$, for which the following holds for $j = 1, ..., m$:

$$\hat{f}_{j,p}(t) \in \underset{f \in F \setminus (F_p^{ue}(t) \cup \bigcup_{i=1}^{j-1} \{\hat{f}_{i,p}(t)\})}{\operatorname{argmax}} \hat{d}_{f,p}(t). \tag{5}$$

If the set of files defined by (5) is not unique, again ties are broken arbitrarily. After selecting the files to be cached, the user demands for these files in this period are observed. Then, the estimated demands and the counters for cached files are updated.

[2]The set $\mathcal{E}_{f,p}(t)$ does not have to be stored since the estimated demand $\hat{d}_{f,p}(t)$ can be updated based on $\hat{d}_{f,p}(t-1)$ and on the observed demand at time t.

IV. Analysis of the Regret

In this section, we give a regret bound for the proposed learning algorithm, where regret describes the loss incurred by the algorithm. We will prove that the regret is sublinear, i.e., $R(T) = O(T^\gamma)$ with $\gamma < 1$. This bound on the regret guarantees that for $T \to \infty$, the algorithm converges to the benchmark solution given by (1) in the sense that the average number of cache hits is maximized in the limit.

The regret bound is enabled by the natural assumption that expected demands for files are similar for similar contexts. For example, at a similar time of day and with similar characteristics of users in the vicinity, the requests for files will also be similar. Such an assumption is crucial to learn from previous observations and can be captured by the following Hölder condition.[3]

Assumption 1. *There exists $L > 0$, $\alpha > 0$ such that for all $f \in F$ and for all $x, y \in \mathcal{X}$, it holds that*

$$|\mu_f(x) - \mu_f(y)| \leq L||x - y||^\alpha,$$

where $|| \cdot ||$ denotes the Euclidian norm in $\mathbb{R}^D$.

The theorem given below shows that the regret of our proposed algorithm m-CLUP is sublinear in the time horizon T.

Theorem 1 (Bound for $R(T)$)**.** *Let $K(t) = t^{\frac{2\alpha}{3\alpha+D}} \log(t)$ and $s_T = \lceil T^{\frac{1}{3\alpha+D}} \rceil$. If m-CLUP is run with these parameters and Assumption 1 holds true, the leading order of the regret is $O\left(mU_{\max}|F|T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.*

The interested reader can find the proof of Theorem 1 in our online appendix [14].

V. Numerical Results

In this section, we numerically evaluate the proposed learning algorithm m-CLUP by comparing its solution to several reference algorithms under different scenarios.

A. Reference Algorithms

We compare m-CLUP with five reference algorithms. The first algorithm is the *optimum benchmark*. This algorithm works under the assumption of complete knowledge, i.e., the expected demands of all files are known a priori. Given context vector $x \in \mathcal{X}$, the optimal benchmark algorithm selects the m files with highest to m-th highest expected demands for this context x as given by (1).

The second reference algorithm is called m-UCB, which consists of a variant of the UCB algorithm. UCB is a classical learning algorithm for multi-armed bandit problems [15], which has logarithmic regret order. However, it does not take into account context information, i.e., the logarithmic regret is with respect to the average expected demand over the whole context space. While in classical UCB, one action is taken in each period, we modify UCB to take m actions at a time, which corresponds to selecting m files.

[3]This assumption is only needed for the analysis of the regret. m-CLUP can be applied even when this assumption does not hold true.

The third reference algorithm is the m-ϵ-Greedy. This is a variant of the simple ϵ-Greedy [15] algorithm, which does not consider context information. This algorithm caches a random set of m files with probability $\epsilon \in (0,1)$. With probability $(1-\epsilon)$, the algorithm caches the m files with highest to m-th highest estimated demands. These estimated demands are calculated based on previous demands for cached files.

The forth reference algorithm is called m-Myopic. This is an algorithm taken from [5], which is investigated since it is comparable to the well known Least Recently Used algorithm (LRU) for caching. m-Myopic learns only from one period in the past. It starts with a random set of files and in each of the following periods discards all files that have not been requested in the previous period. Then it replaces the discarded files randomly by other files.

The fifth reference algorithm is called Random. It is a lower benchmark for the other algorithms, since it caches a random subset of files in each period.

B. Performance Measures

The following performance measures are used in our analysis. One measure is the *aggregated number of cache hits*, which allows comparing the absolute performance of the algorithms. A relative performance measure is given by the *cache efficiency*, which is defined as the ratio of cache hits compared to the overall demand, i.e.,

$$\text{cache efficiency in } \% = \frac{\text{cache hits}}{\text{cache hits} + \text{cache misses}} \cdot 100.$$

The cache efficiency describes the percentage of user requests which can be served by the files cached at the sBS. Note that in general, even the optimum benchmark cannot serve all user requests since first, the cache size is limited and second, the optimum benchmark algorithm has knowledge about the expected demands, but it does not know the actual demand in the future period. Another important performance measure is given by the *average numerical regret* of an algorithm. It is given as the time-averaged difference between the number of realized cache hits in the optimum benchmark and the number of realized cache hits obtained by the learning algorithm.

C. Simulation Model for Context-Dependent Demands

The proposed learning algorithm does not make any assumptions on the underlying distribution of the context-dependent demands. However, to evaluate the performance of the algorithm, we have to model context-dependent demands of files. The set of expected demands $\{\mu_f\}_{f \in F}$ of all files is called popularity profile. As confirmed by measurement studies [16], the popularity profile can be modeled by a Zipf-like distribution that is characterized by a parameter β. In this distribution, given U requests for files from a set of $|F|$ files, the expected demand of the i-th most popular file is given by

$$\mu_i = U \cdot \frac{\frac{1}{i^\beta}}{\Omega},$$

where

$$\Omega = \sum_{j=1}^{|F|} \frac{1}{j^\beta}.$$

The parameter $\beta \geq 0$ accounts for the skewness of the distribution, where a larger β stands for a more skewed distribution. The Zipf-like distribution gives a fixed popularity profile. In order to additionally include context information, for our simulations, we model context-dependent demands of files by adapting the popularity profile for different contexts. We assume that in each period, the sBS observes a 3-dimensional context vector related to the users currently present in its vicinity, i.e., the dimension of the context space is $D = 3$. These three dimensions are (i) the number of users normalized to a maximum number $U_{\max}$, where $U_{\max}$ is the maximum number of users that can be served by the sBS, (ii) the fraction of female users and (iii) the fraction of underage users. We assume that the number of users changes U and thereby the scaling of the expected demands in the distribution above. Further we assume that the fraction of female users and the fraction of underage users changes the order of the popularity among the files. We define 4 user contexts, which are classified based on whether the fraction of female users is below or above 50% and whether the fraction of underage users is below or above 50%. Each of the 4 user contexts has its specific order of files in the popularity ranking. To model this, the set of files F is divided into 4 file classes of size $|F|/4$. Then each of the 4 user contexts is mapped to a specific order of the 4 file classes in the popularity ranking. The popularity profile is adapted in each of the 4 user contexts based on this specific order of the popularity ranking.

D. Results

In our simulations, in m-ϵ-Greedy we set $\epsilon = 0.25$ and in m-CLUP we set the control function to $K(t) = c \cdot t^{\frac{2\alpha}{3\alpha + D}} \log(t)$ with $c = 1/(|F|D)$. Tuning $K(t)$ with $c < 1$, compared to Theorem 1, the number of exploration phases can be reduced.

To evaluate the long-term behavior of our proposed algorithm, we first investigate the following scenario. We assume that the sBS can store $m = 5$ files out of $|F| = 100$ available files, i.e., the cache size corresponds to 5% of the overall file set [5]. In each period, a random number of users is in the vicinity of the sBS. The number of users is uniformly distributed between 20 and $U_{\max} = 50$. The fractions of female and underage users among the users are uniformly distributed between 0 and 1. Note that uniformly distributed context arrivals are among the most difficult scenarios for our algorithm since they require the algorithm to explore all sets in its constructed partition $\mathcal{P}_T$ equally. Hence, we evaluate our algorithm under worst-case conditions. In each period, the file demands are sampled from a context-dependent Zipf-like distribution as described above. The skewness parameter is set to $\beta = 0.7$ (compare measurement results in [16]). The time horizon is set to $T = 10000$. We run all algorithms for this scenario and average the results over 100 realizations of random contexts and demands.

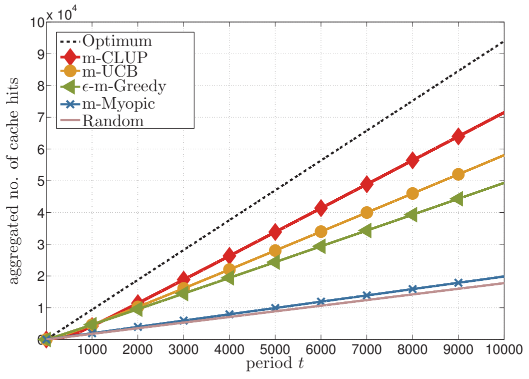

Fig. 2. Aggregated number of cache hits as a function of time

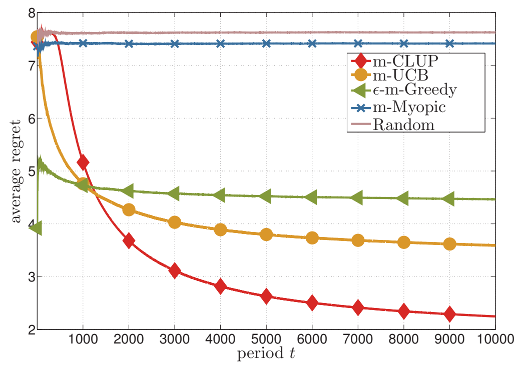

Fig. 3. Average regret as a function of time

The long-term behavior is investigated by plotting the results of the algorithms as a function of time, i.e., over the periods $t = 1, ..., T$. Figure 2 shows the aggregated number of cache hits up to period t as a function of time. Figure 3 shows the average regret up to period t as a function of time. As can be seen from the plots, the optimum benchmark, assuming complete knowledge, serves as upper bound for all other algorithms. In the starting phase, the proposed algorithm m-CLUP shows worse performance than the reference algorithms in terms of aggregated cache hits and average regret. This is due to the cold start in which m-CLUP requires many exploration steps for each set in its partition. After a number of periods, among the non-optimal algorithms, m-CLUP, m-UCB and m-ϵ-Greedy have a much better performance than m-Myopic and Random, since they learn from the whole history of observed demands. However, m-CLUP outperforms all other non-optimal algorithms since it additionally exploits context information already collected, even though the context arrivals are uniformly distributed so that contextual learning is difficult. For all five non-optimal algorithms the aggregated number of cache hits seems to be linearly increasing with smaller slope than that of the optimum benchmark. However, Figure 3 reveals that the average regret of both m-CLUP and m-UCB is decreasing over time, while the one of m-ϵ-Greedy is nearly constant. m-Myopic and Random have a high constant average regret.

Next, we investigate the impact of the cache size m. For this scenario, we set the time horizon to $T = 5000$ and consider a smaller number $|F| = 20$ of files. Then we vary the cache size m between 1 and 20. All remaining parameters are kept as in the first scenario. For varying cache size, Figures 4 and 5 show the cache efficiency in percent and the average regret after $T = 5000$ periods, respectively. For all algorithms, the cache efficiency is increasing for increasing cache size until hitting 100% when all files can be cached, i.e., $m = |F|$. Interestingly, however, the average regret of all non-optimal algorithms is highest when the cache size is in the region of $15-30\%$ of the total file number. Moreover, the results indicate that again m-CLUP and m-UCB slightly outperform m-ϵ-Greedy and clearly outperform m-Myopic and Random. The performance

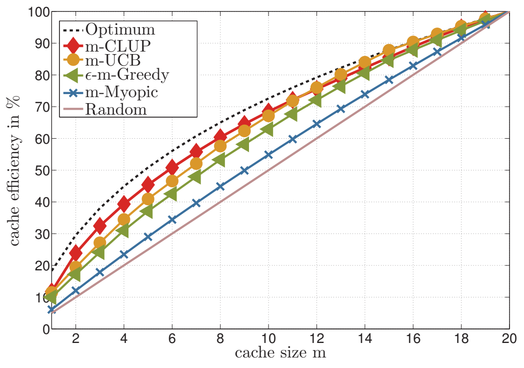

Fig. 4. Cache efficiency in $\%$ for $|F| = 20$ vs. cache size m

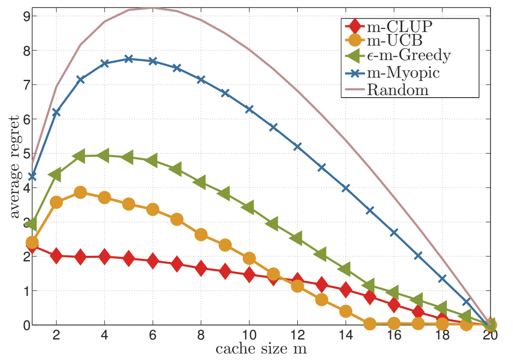

Fig. 5. Average regret for $|F| = 20$ vs. cache size m

of m-UCB is better than that of m-CLUP for cache sizes above 50% of the total file number. For smaller cache sizes, however, m-CLUP has a better performance, which is the practically relevant case, since caches can usually store only a small fraction of all available content.

Finally, the effect of the popularity skew β is studied. Again we set $T = 5000$ and $|F| = 20$. Then we vary the popularity skewness β between 0 and 3. All remaining parameters are kept as in the first scenario. For varying cache size, Figures 6 and 7 show the cache efficiency in percent

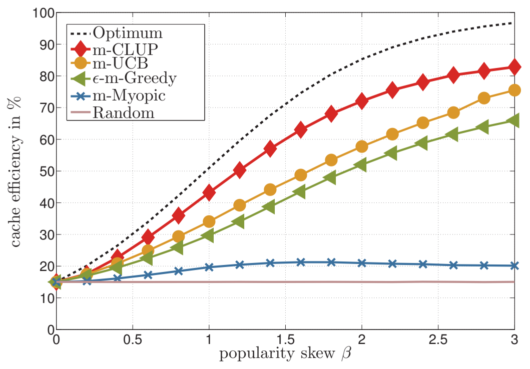

Fig. 6. Cache efficiency in % for $|F| = 20$ vs. popularity skew β

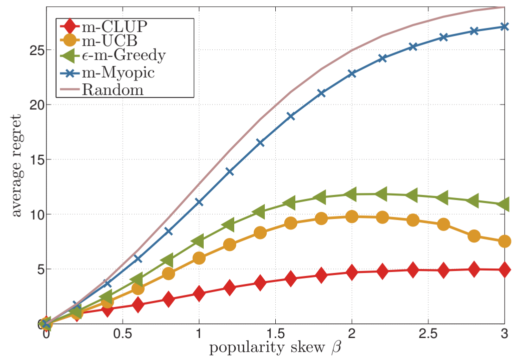

Fig. 7. Average regret for $|F| = 20$ vs. popularity skew β

and the average regret after $T = 5000$ periods, respectively. For increasing popularity skew β, the cache efficiency of all algorithms is increasing. This can be explained by the higher skewness of the popularity distribution for increasing values of β. For high β, only few files are very popular so that a small cache size is sufficient to serve a high number of requests. The average regret of m-CLUP, m-UCB and m-ϵ-Greedy stagnates or decreases for higher β, while it keeps increasing for m-Myopic and Random. Hence, especially in case of highly skewed distributions, a learning algorithm is required for good performance. Apart from very small values of β, m-CLUP yields better results than all other non-optimal algorithms.

VI. Conclusion

In this paper, we investigated smart content caching in wireless small cell networks, in which popular content is stored at an sBS. To maximize the number of cache hits, the sBS should cache the most popular files. We assumed that the popularity profile is (i) varying, for example depending on users in the vicinity of the sBS and (ii) not known a priori. We presented an algorithm based on contextual multi-armed bandits, in which the sBS regularly updates its cache content and observes the demands for cached files. Over time, the sBS then learns the context-dependent popularity profiles. We derived a sublinear regret bound, which proves that our algorithm learns smart caching. Numerical results confirmed that by exploiting contextual information, our proposed algorithm performs well in comparison to reference algorithms in different scenarios.

Acknowledgment

The work by Sabrina Müller and Anja Klein has been funded by the German Research Foundation (DFG) as part of project B03 within the Collaborative Research Center (CRC) 1053 – MAKI. The work by Onur Atan and Mihaela van der Schaar is supported by a DDDAS Airforce grant.

References

[1] [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf

[2] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[3] N. Golrezaei, K. Shanmugam, A. Dimakis, A. Molisch, and G. Caire, "FemtoCaching: Wireless video content delivery through distributed caching helpers," in *Proc. IEEE INFOCOM*, 2012, pp. 1107–1115.

[4] K. Poularakis and L. Tassiulas, "Exploiting user mobility for wireless content delivery," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2013, pp. 1017–1021.

[5] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *IEEE International Conference on Communications (ICC)*, 2014, pp. 1897–1903.

[6] ——, "Multi-armed bandit optimization of cache content in wireless infostation networks," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2014, pp. 51–55.

[7] ——, "Content-level selective offloading in heterogeneous networks: Multi-armed bandit optimization and regret bounds," *arXiv preprint, arXiv: 1407.6154*, 2014.

[8] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[9] ——, "Anticipatory caching in small cell networks: A transfer learning approach," in *1st KuVS Workshop on Anticipatory Networks*, 2014.

[10] M. ElBamby, M. Bennis, W. Saad, and M. Latva-aho, "Content-aware user clustering and caching in wireless small cell networks," in *Proc. IEEE International Symposium on Wireless Communications Systems (ISWCS)*, 2014, pp. 945–949.

[11] T. Lu, D. Pal, and M. Pal, "Contextual multi-armed bandits," in *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 485–492.

[12] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. ACM 19th International Conference on World Wide Web (WWW'10)*, 2010, pp. 661–670.

[13] C. Tekin and M. van der Schaar, "Distributed online learning via cooperative contextual bandits," *IEEE Transactions on Signal Processing*, vol. 63, no. 14, pp. 3700–3714, 2015.

[14] [Online]. Available: http://kang.nt.e-technik.tu-darmstadt.de/nt/fileadmin/kt/Publikationen_PDFs/2016/ICC/ICC_Mueller_App.pdf

[15] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, May 2002.

[16] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," in *Proc. IEEE INFOCOM*, vol. 1, 1999, pp. 126–134.

[17] E. Chlebus, "An approximate formula for a partial sum of the divergent p-series," *Applied Mathematics Letters*, vol. 22, no. 5, pp. 732 – 737, 2009.

[18] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.