Learning to Cooperate in D2D Caching Networks

Georgios S. Paschos, Apostolos Destounis, George Iosifidis

Abstract—We consider a wireless device-to-device (D2D) cooperative network where memory-endowed nodes store and exchange content. Each node generates random file requests following an unknown and possibly arbitrary spatio-temporal process, and a base station (BS) delivers any file that is not found at its neighbors’ cache, at the expense of higher cost. We design an online learning algorithm which minimizes the aggregate delivery cost by assisting each node to decide which files to cache and which files to fetch from the BS and other devices. Our policy relies on the online gradient descent algorithm, is amenable to distributed execution, and achieves asymptotically optimal performance for any request pattern, without prior information.

I. INTRODUCTION

The rapidly growing demand for mobile content delivery [1] creates new revenue opportunities for wireless networks, but also requires to increase rapidly their capacity. Unfortunately, typical solutions based on PHY-layer advances or network densification are constantly outpaced by the increasing demand [19], and this calls for new content delivery approaches. To this end, a potentially game-changing idea is to employ device-to-device (D2D) communications where memory-endowed user devices can cache popular files and exchange them with each other upon request [9]. Such cooperative D2D swarms can increase the network content delivery capacity, mitigate cellular congestion, and improve the end-user experience.

A key challenge in D2D cooperative caching is to design the caching policy, i.e., identify which files to cache at each device at any given time [20]. On the one hand the devices have small storage and therefore can store only a small subset of the possible content files; on the other hand each of them “sees” a small number of requests per unit time and hence estimating the popular files at each location becomes very challenging. In view of these limitations, the devices may consistently fail to store in their cache the files that will be requested in the future by their neighbors, rendering the D2D cache-hit ratio practically negligible and hence this solution ineffective.

Caching policies often assume that requests are generated by a given stationary process, cf. [19], and systems in practice rely on reactive policies such as LFU and LRU. These, however, solve the 1-cache problem conditionally on the request process. For example, LFU is suitable for stationary requests [7], and LRU for the adversarial model [17]. When the actual process is other than assumed these policies perform poorly [18], and this problem is exacerbated in D2D networks where popularity has hot spots in time and space. Recently, [2], [8], [14] proposed dynamic policies for caching networks, e.g., the m-LRU [8] or “lazy rule” [14] policies, which however do not offer performance guarantees. Hence, the problem of designing a policy robust to the D2D network dynamics is an equally challenging and important open problem.

In this cooperative D2D network, users might change positions and content preferences, and hence we need an algorithm that will allow each device to decide: (i) which files to cache for serving its neighbors’ needs; (ii) from which neighbors to retrieve a file, and when to use the last-resort solution of the base station (BS), Fig. 1. This requires a learning mechanism for making caching and routing decisions in an online and decentralized fashion. Previous efforts employing learning are restricted either to inference of the popularity model [3], [4]; or rely in Q-learning [21], [24] and classification techniques [15] to estimate request frequencies. These approaches are not applicable to this D2D scenario as they are centralized, exhibit often high complexity, presume a stationary model, and do not decide routing. Other interesting suggestions for D2D networks, see [6], [13], suffer from computational complexity or rely on assumptions that are valid only in some cases, e.g., known popularity. Finally, [11] proposes a decentralized D2D file sharing algorithm which however needs access to the pattern of file requests and network evolution.

Here we design a caching policy that has universally-optimal performance, which is defined as the cost for delivering the requested files to users through (cheap) D2D or (costly) BS-to-device transmissions. We formulate the D2D caching operation as an online convex optimization (OCO) problem, and develop a dynamic and distributed algorithm that solves it without the need to make any assumption about the request pattern. That is, our policy ensures asymptotically no regret, as it achieves no more average cost than a static caching configuration selected with knowledge of future requests.

The contributions of this paper can be summarized as follows. (i) We propose the idea of embedding a distributed online learning mechanism to D2D caching policies. We achieve this by formulating an OCO problem, and this opens a link between caching and this novel machine learning tool. (ii) We design an online caching policy that leverages the online gradient descent algorithm to achieve asymptotically optimal performance under any possible spatio-temporal request pat-
tern in our D2D network. We also explain how our policy can adapt to network changes and user churn. (iii) We compare our policy with the state-of-the-art mLRU and “lazy” LRU policies, verifying that it outperforms its competitors, while converging to the optimal static policy.

II. System Model

Network. Consider a set $I$ of $I$ wireless users in an area, each one with a cache of size $C_i > 0$, $i \in I$. Let $\mathcal{L}$ be the set of direct links connecting the users, where a link appears if the devices’ proximity and the electromagnetic environment allows it to be reliably established. A link between $i$ and $j$ is associated to a cost $c_{ij}$, $(i,j) \in \mathcal{L}$, which represents, e.g., application-layer latency performance or energy consumption during content transmission; and we assume $c_{ii} = 0, \forall i \in I$. All users maintain a connection with a base station (BS, subscripted with $i$). The users can obtain any file from the BS at cost $c_{0i} > c_{ij}$, $\forall i \in I, (i,j) \in \mathcal{L}$. We assume that links can deliver the requested content in the considered time window.

Requests. There is a catalog $N$ with $N$ files of unit size. The system operation is time-slotted, and $\ell^{i,n} \in \{0,1\}$ denotes the event that a request for file $n$ has been submitted by user $i$ during slot $t$. At each $t$ we assume there is one request, or, from a different perspective, that the system decisions are updated after each request. Hence, the request process is described by a sequence of vectors $\{r_t\}_{t=1,\ldots,T}$ drawn from set:

$$R = \left\{ r \in \{0,1\}^{NI} \Big| \sum_{n \in N} \sum_{i \in I} r_{i,n} = 1 \right\}. \quad (1)$$

The instantaneous file popularity is expressed by the probability distribution $P(r_t)$ (with support $R$), which is allowed to be unknown and arbitrary. The same holds for the joint distribution $P(r_1,\ldots,r_T)$ that describes the file popularity evolution, for any user location, and within an interval of $T$ slots. This generic model captures all possible spatio-temporal request sequences, including stationary (i.i.d. or otherwise), non-stationary, and adversarial models. The latter is the most general case, as they include request sequences selected by an adversary aiming to disrupt the system performance.

Caching. The cache of each user $i \in I$ can store only $C_i < N$ files, but the BS has the entire catalog. Following the standard practice in wireless caching models, we perform caching using the Maximum Distance Separable (MDS) codes. In MDS, the files are split into a fixed number of $F$ data chunks, and we store in each cache an amount of coded chunks that are pseudo-random linear combinations of the data chunks. Using the MDS properties, a user can decode the file (with high probability) if it receives any $F$ coded chunks. Hence, the caching decision vector $y_t$ has $N \cdot I$ elements, where $y_{t,n} \in [0,1]$ denotes the amount of random coded chunks of file $n$ stored at user $i$ during slot $t$.

Based on this, we introduce the convex set of eligible caching vectors:

$$\mathcal{Y} = \left\{ y \in [0,1]^N \Big| \sum_{n \in N} y_{i,n} \leq C_i, \; i \in I \right\}. \quad (2)$$

We are interested in distributed policies, where each user $i$ changes its cache $y_t^i = (y_{t,n}^i, n \in N)$ based on information from its one-hop neighbors $j : (i,j) \in \mathcal{L}$. Thus, we define:

**Definition 1** (Local Caching Policy). A local caching policy $\sigma^i$ for user $i$ is a (possibly randomized) rule

$$\sigma^i (r_1,\ldots,r_{t-1};y_1^i,\ldots,y_{t-1}^i) :\rightarrow y_t^i \in [0,1]^N.$$ 

The collection $\sigma = \{\sigma^i, i \in I\}$ of the caching policies for all users will be henceforth referred to as a “caching policy”.

Routing. Since each user $i \in I$ might have more than one neighbors, we introduce routing variables to determine the cache from which the requested file will be fetched. Let $z_{t,n}^{i,j} \in [0,1]$ denote the portion of request $r_{t,n}^i$ that is fetched from cache $j$, and we define the routing vector $z_t = (z_{t,n}^{i,j}, n \in N, i,j \in I)$ implemented in slot $t$. There are two important remarks here. First, due to the coded caching model, the requests can be simultaneously routed from multiple caches. In terms of communications, this can be implemented through time-sharing among the activated links, or using concurrently different network interfaces. Second, the caching and routing decisions are coupled and constrained: (i) a request cannot be routed from an unreachable cache, (ii) we cannot route from a cache more data chunks than it has, and (iii) each request must be fully routed.

We define $\ell_{ij} = 1$ if $\{i,j\} \in \mathcal{L}$ and $\ell_{ij} = 0$ otherwise, and thus the set of eligible routing decisions conditioned on $y_t$ is:

$$\mathcal{Z}_t (y_t) = \left\{ z \in [0,1]^{N^2} \Big| \sum_{j \in \mathcal{L}_+} z_{t,n}^{i,j} \leq r_{t,n}^i, \; z_{t,n}^{i,j} \leq \ell_{ij} y_{j,n}, \; i,j \in \mathcal{I}, n \in \mathcal{N} \right\}. \quad (3)$$

Note that $z_{t,n,0}^{i,0}$ does not appear in the second constraint, because the BS stores the entire catalog and can serve all users. This last-resort routing option ensures that $\mathcal{Z}_t (y_t)$ is non-empty for any $y_t \in \mathcal{Y}$. As it will become clear next, the optimal routing decisions can be devised for a given cache configuration. This is an inherent property of link-uncapacitated caching networks, see also \cite{9,19}.

III. Problem Formulation

A file request of a node can be served, exclusively or partially (due to MDS), by neighboring devices at a smaller cost than fetching it from the base station. Given a cache configuration $y_t$, the (minimum) cost to satisfy $r_t$ is:

$$f_t (y_t) = \min_{z_t \in \mathcal{Z}_t (y_t)} \sum_{n \in \mathcal{N}} \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{L}_+} r_{t,n}^i c_{ij} z_{t,n}^{i,j}, \quad (2)$$

where the optimization decides the routing that minimizes the cost for a given file placement at the nodes. The function’s coded chunks is supported by the observation that large files are composed of thousands chunks, stored independently, see literature of partial caching \cite{16}. Hence, by rounding these fine-grained fractional decisions, we will only induce a small application-specific error. In prior caching models, fractional variables represent probabilities of caching \cite{5,22}.1
knowledge of all requests in the time horizon of interest.
Hence, the regret of policy configuration in hindsight. In our case, this benchmark is the performance metric of an algorithm in this line of work is the Convex Optimization framework [22]. The performance metric suggests that is beneficial if the file has been cached at time $t$. Lemma 1.

Proof: Fix a request vector $r_t$, and consider cache configurations $y_1, y_2 \in \mathcal{Y}$; note that, for any $\lambda \in [0, 1]$, $\mathcal{Y} = \lambda y_1 + (1-\lambda) y_2$ is also a valid configuration. We will show that:

$$f_t(\lambda y_1 + (1 - \lambda) y_2) \leq \lambda f_t(y_1) + (1 - \lambda) f_t(y_2).$$

Let us denote $z_1^*, z_2^*, z^*$ the optimal routing vectors corresponding to $y_1, y_2, \mathcal{Y}$, respectively. We then have:

$$f_t(y_k) = \sum_{n \in N} \sum_{j \in J_k} r_{i,n}^t c_{ij} z_{i,n,j}^*, \forall k \in \{1, 2\}, \text{ and (3)}$$

$$f_t(\mathcal{Y}) = \sum_{n \in N} \sum_{j \in J_k} r_{i,n}^t c_{ij} z_{i,n,j}^*, \forall z \in Z_t(\mathcal{Y}).$$

It holds $\lambda z_1^* + (1 - \lambda) z_2^* \in Z_t(\mathcal{Y})$, thus $f_t(\mathcal{Y}) \leq \lambda f_t(y_1) + (1 - \lambda) f_t(y_2) = \sum_{n \in N} \sum_{j \in J_k} r_{i,n}^t c_{ij} \left(\lambda z_{i,n,j}^* + (1 - \lambda) z_2^*\right)$. We then have:

$$\sum_{n \in N} \sum_{j \in J_k} r_{i,n}^t c_{ij} \left(\lambda z_{i,n,j}^* + (1 - \lambda) z_2^*\right) \leq f_t(\lambda y_1 + (1 - \lambda) y_2).$$

Subscript $t$ at the cost function [2] reminds us its dependence on the request $r_t$ that is generated at $t$. Since these events may vary according to a non-stationary process, we will use the concept of regret from online convex optimization [22].

We capture that the request sequence may follow any arbitrary and a priori unknown probability distribution, by using the idea of an adversary which selects $r_t$ at each slot $t$, while knowing $y_t$. This assumption reflects that, in practice, caches are populated before the requests are issued. Since by Lemma 1, $f_t(y)$ are convex, our problem falls in the Online Convex Optimization framework [22]. The performance metric of an algorithm in this line of work is the regret: the difference between costs incurred by the algorithm and the best static configuration in hindsight. In our case, this benchmark is the optimal cache configuration (same for all slots) devised with knowledge of all requests in the time horizon of interest $T$. Hence, the regret of policy $\sigma$ is:

$$R_T(\sigma) = \max_{P(r_1, r_2, ..., r_T)} \mathbb{E} \left[ \sum_{t=1}^{T} f_t(y_t(\sigma)) - \sum_{t=1}^{T} f_t(y_t^*) \right].$$

The expectation is over the joint probability distribution of requests and possible randomizations in $\sigma$ and,

$$y^* \in arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} f_t(y),$$

is the best fixed action in hindsight, i.e. the best chunk placement over the entire sample path of requests. Our goal is to devise a policy whose regret scales sublinearly with $T$:

$$\lim_{T \to \infty} R_T(\sigma)/T = 0.$$
Then \( g_t(y_t) \) is a subgradient of \( f_t \) at \( y_t \), that is: \( f_t(y') \geq f_t(y_t) + g_t(y_t) \top (y' - y_t), \forall y' \in \mathcal{Y} \).

**Proof.** We start by denoting \( \beta^*(y') \) the outcome of (10) for cache configuration \( y' \in \mathcal{Y} \) and define the function:

\[
\Lambda(y, \beta) = \max_{\alpha \in \mathbb{R}} \min_{z \in [0, 1]^I} L(y, z, \alpha, \beta), \forall y \in \mathcal{Y}, \text{ and hence:}
\]

\[
f_t(y_t) = \min_{\alpha \in \mathbb{R}} \max_{z \in [0, 1]^I} L(y_t, z, \alpha, \beta^*(y_t)) = \Lambda(y_t, \beta^*(y_t)) = \Lambda(y_t, \beta^*(y_t)) + \beta^*(y_t)(y_t - y_t) \leq \Lambda(y_t, \beta^*(y_t)) + \beta^*(y_t)(y_t - y_t),
\]

where (a) holds since [6]-[8] has the strong duality property; (b) holds since \( \Lambda() \) is linear and we can maximize successively over the different primal or dual variables; and (c) holds as only \( -\sum_{j \in \mathcal{J}(u_t)} \beta_j y_j \top \mathbb{I}_{L_t} \) on \( y_t \). Due to strong duality for \( y_t \), we can replace \( f_t(y') = \Lambda(y', \beta^*(y')) \), and then sufﬁces to rearrange terms.

Since \( \beta^*_t(y_t) \) is the optimal multiplier for (8), it has nonzero elements only where this constraint is tight. Intuitively, this means that after user \( i_t \) requests file \( n_t \), the direction of the subgradient is towards caching more parts of this ﬁle at user \( i_t \) and at users having low-cost D2D links with \( i_t \).

**B. Algorithm Design**

The Distributed Online Caching Policy (DOCP) is shown in Algorithm 1. The execution of the policy is iterative, where in each slot \( t \) the following steps take place. First, a user \( i_t \) submits a request for a file \( n_t \) (step 3). This user solves (6)-(7) to ﬁnd \( z_t^e = \{ z_{n_t}^e, r_{i_t}^e | e \in J \} \), and requests the parts of \( n_t \) from the respective neighbors or the BS (step 5). A certain utility is obtained based on previous requests. Then, user \( i_t \) sends the optimal multiplier \( \beta^*_t(y_t) \) to each neighbor \( j \in \mathcal{J}(i_t) \) (step 8) which updates its caching policy accordingly. This involves calculating the new \( y_{t+1} \), based on the latest request, and projecting them back into the feasible space (step 9).

\[
y_{t+1} = \Pi_{[0, 1]^I} (y_t + \gamma \beta^*_t(y_t) e_{n_t}),
\]

where \( \Pi_{\mathcal{X}}(.) \) is the Euclidean projection on \( \mathcal{X} \), and \( e_n \in \mathbb{R}^N \) has zero elements except the \( n \)-th element being equal to 1.

Note that DOCP is indeed distributed, since only the neighbors of each requestor need to update their caches. Moreover, this update is based solely on messages received by the requester, and these communication overheads are moderate as only the Lagrange multipliers are sent to 1-hop neighbors. Finally, the projection operation can be executed efﬁciently, i.e., in \( O(N \log N) \) runtime, and for each user independently, by using the local projection algorithm introduced in [18]. We omit the details here due to lack of space.

**C. Performance Guarantees**

The next theorem proves that DOCP achieves no regret performance, under any possible spatio-temporal arrival pattern.

**Theorem 1** (Regret of DOCP). For step size \( \gamma = \frac{\sqrt{2CJ}}{c \sqrt{T}} \), the regret of DOCP satisfies:

\[
R_T(\text{DOCP}) \leq c^* \sqrt{2CJ \sqrt{T}},
\]

where we have replaced the previously constant link costs with slot-speciﬁc ones \( c_{i,j} = \{ c_{i,j} | (i, j) \in \mathcal{L}, t = 1, \ldots, T \} \). For instance, if link \( (i, j) \) exists in slot \( t \) but not in slot \( t+1 \) (e.g., nodes have moved farther), then we can use \( c_{i,j} = c_{\max} > c_{i,j} \forall i \in \mathcal{I} \) which will make this link non-eligible for DOCP (the BS is available and cheaper).

**Algorithm 1: Distributed Online Caching Policy (DOCP)**

1. **Input:** Select step size \( \gamma \)
2. for \( t = 1, 2, \ldots, T \) do
3. User making request: \( i_t \); ﬁle requested: \( n_t \).
4. User \( i_t \) solves (6) - (8) to ﬁnd \( z_t^e = \{ z_{n_t}^e, r_{i_t}^e | e \in J \} \).
5. User \( i_t \) fetches data from BS or D2D links, based on \( z_t^e \).
6. Utility \( f_t \) is accrued based on \( y_t^e, z_t^e \).
7. New caching conﬁguration is devised:
   - for \( j = 1, \ldots, |J(i_t)| \) do
     - User \( i_t \) sends to user \( j \) message \( \beta^*_j \) using (10).
     - User \( j \) updates its cache by \( y_{n_t,i}^e = \Pi_{[0, 1]^N} (z_{n_t,i}^e) \).
   - end
8. end
   - end

where, we deﬁned the parameters \( c^* = \max_{i \in I} c_{i,0} \), \( C = \max_{i \in I} C_i \), and \( J^* = \max_{i \in I} |J(i)| \).

**Proof:** Using non-expansiveness of Euclidean projection:

\[
\|y_{t+1} - y_t\|^2 \leq \|y_t - y_t^*\|^2 + \gamma^2 \sum_{t=1}^T \|g_t(y_t)\|^2 - 2\gamma \|y_t^*\|(y_t - y_t^*)
\]

Also, a telescopic sum over \( T \) slots gives \( \|y_T - y^*\|^2 \leq \|y_1 - y^*\|^2 + \gamma^2 \sum_{t=1}^T \|g_t(y_t)\|^2 - 2\gamma \|y_T^*\| \leq 2CJ^* \leq 2\gamma \|y_T - y^*\|^2 \).

To proceed, note that \( \|y_{t+1} - y_t\|^2 \leq 2CJ^* \) and \( \|g_t(y_t)\|^2 \leq (\max_{i \in I} C_i) \).

Using that \( \|y_T - y^*\|^2 \leq 2CJ^* \leq 2\gamma \|y_T - y^*\|^2 \).

Furthermore, due to convexity of \( f_t(y) \), it holds \( f_t(y^*) - f_t(y) \geq g_t(y) \top (y^* - y) \), thus:

\[
R_T(\text{DOCP}) = \sum_{t=1}^T (f_t(y_{t+1}) - f_t(y_t)) \leq 2CJ^* + \gamma T(c^*)^2.
\]

The value of the step size and regret bound follow by minimizing the Right Hand Side of the above inequality.
is interesting to note that this extension does not change the regret bound, which is set by the highest cost of the available links, that remains the one between any device and the BS.

V. NUMERICAL RESULTS

We illustrate the performance of DOCP in a setting with $N = 100$ files, and 8 devices which are equipped with a cache of capacity $C = 6$ and are placed randomly in a cell of size 1.5Km. Devices can communicate if they are within a range of 500m, as in current LTE-direct standards [10]. The (relative) cost of downloading a file from the base station is set to 10; a device can fetch a file from its cache at no cost; and the respective costs from other devices vary with the distance: 2 if the device is closer that 100m, 5 if the distance is within $[100m, 300m)$, 7 if in $[300m, 400m)$, and 9 if in $[400m, 500m]$. File requests are drawn from a power law distribution with exponent 0.9. We compare DOCP with the best static policy in hindsight and the lazy LRU and mLRU.

Our results are presented in Fig. 2 which shows the empirical average of the cost for the different policies. We observe that DOCP outperforms both LRU and mLRU, and the margin gets wider as time progresses. In addition, the performance of DOCP gets closer to the one of the best policy in hindsight, thus verifying the no-regret theoretical guarantee. Figure 3 compares the total cache allocation, i.e., the total fraction of each file cached at the devices, for DOCP and the best static hindsight policy. We see that, while the algorithm starts from an almost uniform allocation, by the end of the time interval $T = 4000$ the DOCP cache contents are very aligned with the best configuration in hindsight. This demonstrates that DOCP indeed tends to learn the best static configuration.

VI. CONCLUSIONS

D2D cooperative caching is certainly very promising, but raises previously unseen challenges in devising effective caching policies. Here, we used OCO, a fast-developing area of machine learning, to design an online distributed caching and routing policy that adapts to any (unknown) spatio-temporal request process. This makes it an ideal candidate for such dynamic, often sparse, caching networks. Our work opens a new exciting area at the nexus of online learning and D2D caching systems, and a fascinating next step is to explore how such mechanisms can incorporate incentives for ensuring users’ cooperation, leveraging credit mechanisms [12] or the human tendency to build reciprocal sharing relationships [23].

REFERENCES

[1] Cisco visual networking index: Global mobile data traffic forecast update 2017–2022. White Paper, 2015.
[2] K. Avrachenkov, J. Goseling, and B. Serbeci. A low-complexity approach to distributed cooperative caching with geographic constraints. Proc. of ACM Meas. Anal. Computing Systems, 1(1):1–827, 2017.
[3] E. Baquè et al. A transfer learning approach for cache-enabled wireless networks. In Proc. of Dyntcom, May 2015.
[4] B. N. Bharath, K. G. Nagananda, and V. H. Poor. A learning-based approach to caching in heterogeneous small cell networks. IEEE Trans. on Communications, 64(4), 2016.
[5] B. Blaszczyszyn and A. Giovanidis. Optimal geographic caching in cellular networks. arXiv:1409.7626, 2014.
[6] B. Chen and C. Yang. Caching policy for cache-enabled d2d communications by learning user preference. IEEE Trans. on Communications, 66(12), 2018.
[7] C. Fricker, P. Robert, and J. Roberts. A versatile and accurate approximation for LRU cache performance. In ICT, 2012.
[8] A. Giovanidis and A. Avrancas. Spatial multi-LRU: Distributed caching for wireless networks with coverage overlaps. arXiv:1612.04363, 2016.
[9] N. Golrezaei et al. Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution. IEEE Communications Magazine, 51(4):142–149, April 2013.
[10] M. Haus, M. Waqas, A. Y. Ding, Y. Li, S. Tarkoma, and J. Ott. Security and privacy in device-to-device (d2d) communication: A review. IEEE Communications Surveys Tutorials, 19(2):1054–1079, 2017.
[11] S. Ioannidis, L. Massoulie, and A. Chainteau. Distributed caching over heterogeneous mobile networks. In ACM SIGMETRICS.
[12] G. Iosifidis et al. Efficient and fair collaborative mobile internet access. IEEE/ACM Trans. on Networking, 25(3):20–27, 2017.
[13] W. Jiang, G. Feng, S. Qin, T. Yum, and G. Cao. Multi-agent reinforcement learning for efficient content caching in mobile d2d networks. IEEE Trans. on Wireless Communications, to appear, 2019.
[14] E. Leonardi and G. Neglia. Implicit coordination of caches in small cell networks under unknown popularity profiles. IEEE JSAC, 36(6), 2018.
[15] S. Li et al. Trend-aware video caching through online learning. IEEE Trans. Multimedia, 18(12), 2016.
[16] L. Maggi et al. Adapting caching to audience retention rate. Comp. Comm., 116, 2018.
[17] R. L. Mattson et al. Evaluation techniques for storage hierarchies. In IBM Systems Journal, 1970.
[18] G. Paschos, A. Destounis, L. Vignieri, and G. Iosifidis. Learning to cache with no regret. In IEEE INFOCOM, 2019.
[19] G. Paschos et al. The role of caching in future communication systems and networks. IEEE JSAC, 36(6), 2018.
[20] G. S. Paschos et al. Wireless caching: Technical misconceptions and business barriers. IEEE Commun. Mag., 54(6), 2016.
[21] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis. Optimal and scalable caching for 5G using reinforcement learning of space-time popularities. IEEE J. on Sel. Areas in Sig. Proc., 12(1), 2018.
[22] S. Shalev-Shwartz. Online Learning and Online Convex Optimization. Now Publishers Inc., 2012.
[23] H. Shirado, G. Iosifidis, L. Tassiulas, and N. Christakis. Resource sharing in technologically-defined social networks. Nature Comm., 2019.
[24] S. O. Somuwayi et al. A reinforcement-learning approach to proactive caching in wireless networks. IEEE JSAC, 36(6), 2018.
[25] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In ICML, 2003.