Distributed Caching based on Decentralized Learning Automata

Loris Marini, Jun Li, and Yonghui Li
School of Electrical Engineering,
University of Sydney, Sydney, NSW , AUSTRALIA
Email: {loris.marini, jun.li1, yonghui.li}@sydney.edu.au

Abstract—**In this paper we propose a novel distributed caching scheme in Heterogeneous Cellular Networks (HCN). We are interested in optimizing the content placement in order to minimize the downloading latency. We achieve this in a decentralized manner, based on a game of independent learning automata (LA). First, we propose a faster-converging discrete generalist pursuit algorithm (DGPA) for a single LA based on the concept of conditional inaction (CI), referred to as CI-DGPA. Then we develop a framework for a game of LA based on CI-DGPA defining the information exchange between learners and the environment. Within this framework, we design a reward function that approaches the performance of a greedy algorithm and show that a smart partition of the search space can double the game convergence speed, thereby halving the overhead due to signalling. Simulations show that our scheme can approach the greedy algorithm with a very small performance gap while providing a much lower computational complexity.**

I. Introduction

With the rapid increase in the number of mobile user terminals (UT) and vast demand from bandwidth-hungry mobile services, wireless data traffic is expected to increase by three orders of magnitude over the next ten years. Heterogeneous cellular networks (HCN) has been shown as a promising approach to increasing the capacity while enhancing coverage of cellular networks. In HCNs, low power nodes, referred to as small-cell base stations (SBS), are flexibly deployed to help offload the traffic from the Macro-cell base stations (MBS) [1].

Among all wireless data traffic, the data traffic of video on demand (VoD) is expected to increase of two orders of magnitude in the next five years [2]. In VoD, the bulk of download requests have been placed over a small portion of popular content, which leads to redundant transmissions and low spectrum efficiency. To mitigate the redundancy of data transmissions and offload the data traffic from the MBS, an efficient solution is to locally store popular data into Helpers, known as caching [3, 4]. These Helpers can help provide data downloading services to their adjacent UTs, by exploiting their storage capacity to offload the downloading of popular contents from MBS and thus improve the spectrum efficiency. In HCNs, the function of Helpers can be performed by SBSs, such as femto-cells and pico-cells base stations (BS).

One prominent advantage of data caching is the reduction of downloading latency due to the short distance communications between Helpers and UTs. How to place the data to the Helpers to minimize the downloading latency is a non-trivial problem. It has been proved in [1] that if the deployment of Helpers is dense enough so that UTs can chose between two or more Helpers, the optimization problem becomes NP-Hard and is intractable. A number of attempts have been made to find a low complexity algorithm able to estimate the optimal data placement in HCNs [1, 5–7].

The authors in [1] proposed a sub-optimal greedy strategy provably within a factor of two of the optimum. In [5] a centralized greedy algorithm is proposed to tackle the problem of multiple-type data offloading under realistic assumptions. However, both of the two papers assume the full channel state information (CSI) at the MBS, which is not very practical in real systems. Moreover, these two centralized algorithms imposed a heavy computational burden on the MBS, which is unfeasible for large networks. In [7], a machine learning algorithm was shown to learn the popularity profile of files, based on the observation of instantaneous demand of cached content. In [6], a distributed algorithm based on machine learning was proposed to minimize the communications cost of cache replacement. However, both [6] and [7] only focused on a single learner scenario, leaving open the question of a multy-agent implementation.

In this paper we propose a distributed and decentralized algorithm to optimize the cache content placement in HCNs. The benefit of a decentralized approach is twofold: 1) There is no need for full CSI at the MBS; 2) The computational resources at the MBS do not limit the network caching size. On the other hand, a distributed approach ensures low communication complexity since learners act independently without exchanging information. Learning automata (LA) [8] have been extensively shown to be suitable for solving NP-hard problems [9]. In this study, we aim at coordinating a team of independent LA, so that they collectively converge to the content placement that minimizes the latency perceived by the users. The design of a time-invariant reward function equally suitable to every learner, has presented a number of challenges and it constitutes the central contribution of this work.

First, we introduce the concept of conditional inaction (CI) and propose a faster-converging discrete generalized pursuit algorithm (DGPA) for a single LA, referred to as CI-DGPA. Then, we develop a framework for a game of independent LA by 1) defining a learning environment, 2) specifying how LAs interact with it, 3) designing a suitable reward function. Finally, we show that a smart partition of the search space can double the game convergence speed, thereby halving the overhead due to signaling. Simulations show that our scheme closely approaches the performance of a centralized greedy algorithm (benchmark), without relying on full CSI and ensuring low computational complexity.

II. System Model

Consider a single MBS serving N simultaneously-active user terminals (UTs) represented by the set $\mathcal{U} = \{u_1, \ldots, u_N\}$. Each UT is seeking to download a file from a library of files $\mathcal{F}$ available at the MBS, where $\mathcal{F} = \{f_1, \cdots, f_F\}$. We denote by $\mathbf{l} = [l_1, \cdots, l_F]$ be the vector of files' demand distribution, where l_i, $i = 1, \cdots, F$, is the probability that any UTs place a request over file f_i. Within the cell a set of H Helpers $\mathcal{H} = \{h_1, \cdots, h_H\}$ cache M files each from the library $\mathcal{F}$, and can serve an unlimited number of requests from UTs. The files placement can be described by the $M \times H$ file allocation matrix $\mathbf{A}$ whose element $a_{k,j}$ represents the k-th file cached by Helper h_j with $k = 1, \cdots, M$ and $j = 1, \cdots, H$. At any given time, UTs dispose of a set of content providers composed by the MBS, together with H Helpers scattered across the cell. Let $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \ldots, \Pi_{H+1}\}$ be such a set, where Π_1 represents the MBS, Π_2 is the first Helper h_1, Π_3 is the second Helper h_2 and so on. We assume that UTs chose to download each file from the provider with the highest signal-to-noise ratio (SNR), referred to as highest SNR policy (HSP). A consequence of the HSP is that Helpers are never chosen by any users unless they can provide content with a lower latency than the MBS. In the rest of the paper we will assume that files cannot be fragmented. Moreover the network caching capacity, that is the maximum number of files that can be simultaneously off-loaded from the MBS, equals the size of the library such that $F = H \cdot M$.

The channel gain between UTs and providers can be written as $d_{n,j}^{-\alpha}$, where $d_{n,j}$ represents the line-of-sight distance between u_n and Π_j, and α is the path loss coefficient. Assuming transmissions at the Shannon bound, the downloading latency between UT u_n and provider Π_j can be written as

$$\tau_{n,\Pi_j} = \frac{B}{W \cdot \log(1 + \rho \cdot d_{n,j}^{-\alpha})}, \quad (1)$$

where B is the file length (bits), W the channel bandwidth (Hertz) and ρ the link's SNR. The static wireless network can be fully described by a weighted bipartite graph as shown in Fig. 1. Here an edge (u, π) denotes that a communication link exists from UT u_n to provider π_j with a latency τ_{n,π_j} indicated by its corresponding weight. In order to ensure the Helpers have a chance of being selected by the UTs, the network delays in Eq.(1) are assigned as follows:

$$\tau_{n,\Pi_j} = \begin{cases} \tau_{n,\text{MBS}} & \text{if} \quad j = 1 \\ \tau_{n,h_j} & \text{if} \quad \tau_{n,h_j} < \tau_{n,\Pi_1} \\ \infty & \text{otherwise} \end{cases}.$$

The number of Helpers that can communicate with the n-th UT is denoted by d_{u_n} and referred to as degree of u_n. Similarly, d_{h_j} is the degree of Helper h_j and represents the set of users connected to Helper j. The solution to the caching problem is trivial when each UT can connect only to a single

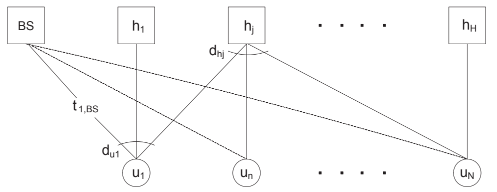

Fig. 1. The Network can be represented by means of a bi-partite graph.

Helper and this corresponds to caching in each Helper the most popular files [1]. However, when users can chose where to download the files, that is $d_{u_n} > 1$, the caching problem becomes NP-hard. In this work we enforce this constraint by considering only networks for which $d_{u_n} \geq 2, \forall u_n \in \mathcal{U}$.

A. Optimal Allocation

Our goal is to determine the allocation matrix $\mathbf{A}$ that minimizes the delay perceived by the UTs when downloading files from a library $\mathcal{F}$. Let $\Pi_j^{(i,n)} \in \mathbf{\Pi}$ denote that the j-th provider can serve file f_i to UT u_n. Let $\tau_{n,\Pi_j^{(i,n)}}$ be the latency of the corresponding link. For each file $f_i \in \mathcal{F}$, UTs apply the HSP to select the best provider $\Pi_{j^*}^{(i,n)} \in \mathbf{\Pi}$ as follows:

$$\Pi_{j^*}^{(i,n)} = \underset{\Pi_j^{(i,n)} \in \mathbf{\Pi}}{\operatorname{argmin}} \left\{ \tau_{n,\Pi_j^{(i,n)}} \right\}, \quad (2)$$

Let $\mathbf{\Gamma}_n = [\Gamma_n^{(1)}, \ldots, \Gamma_n^{(F)}]$ be the $1 \times F$ vector of delays corresponding to the HSP-selected files for the n-th UT, where

$$\Gamma_n^{(i)} = \tau_{n,\Pi_{j^*}^{(i,n)}}. \quad (3)$$

We define the weighted delay (WD) Ω_n for user u_n as

$$\Omega_n = \sum_{i=1}^{F} l_i \cdot \Gamma_n^{(i)}, \quad (4)$$

and the network average weighted delay (NAWD) $\bar{\Omega}$ as

$$\bar{\Omega} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{F} l_i \cdot \Gamma_n^{(i)}. \quad (5)$$

Our goal is to find the allocation matrix to minimize the NAWD. We refer to such allocation $\mathbf{A}_{\text{o}}$ as the optimal solution, denoted by

$$\mathbf{A}_{\text{o}} = \arg\min_{\mathbf{A}} \left\{ \bar{\Omega} \right\}. \quad (6)$$

III. Learning Automata

The goal of a Learning Automata (LA) is to determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded [8]. A LA operates on a set of actions $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_S\}$ and receives a feedback β represented by a binary variable $\beta \in [0, 1]$ resulting from the interaction with an environment, which is the system the automaton learns from. In an iterative fashion, the automaton uses this response and the knowledge acquired in the past actions to determine the next action. A LA operates on the vector $\boldsymbol{P}(t)$ referred to as the action probability vector

$$\boldsymbol{P}(t) = [p_1(t),\ \cdots,\ p_S(t)], \tag{7}$$

where $p_i(t) = Pr[\sigma(t) = \alpha_i] \geq 0$ is the probability that the automaton will select the action α_i at time t and it satisfies

$$\sum_{m=1} p_i(t) = 1 \quad \text{for all "}t\text{"}, \tag{8}$$

and a vector $\hat{\boldsymbol{d}}(t)$ of estimates for the reward probability of the actions set $\mathcal{A}$ is given by

$$\hat{\boldsymbol{d}}(t) = [d_1(t),\ \cdots,\ d_S(t)]. \tag{9}$$

At time t, the LA chooses an action $\sigma(t) \in \mathcal{A}$ according to the probability mass function $\mathbf{P}(t)$ and uses the response from the Environment β and the vector $\hat{\boldsymbol{d}}(t)$ to calculate $\mathbf{P}(t+1)$ and therefore influence the selection of the action in the next iteration.

A. Discrete Generalized Pursuit Algorithm

We consider a type of variable-structure stochastic automata VSSA known as *Pursuit Learning Automata*. Specifically, we focus on discrete generalized pursuit learning Automata (DGPA) as they were empirically proven to be the fastest and most accurate algorithm for Learning Automata [8]. A DGPA generalizes the concept of the pursuit algorithm by pursuing all the actions that, according to the reward probability estimates, are more likely to be rewarded than the current chosen action. The algorithm recursively updates the actions probability vector $\mathbf{P}(t)$ by means of a direction vector $\mathbf{e}(t)$ according to the following equation [8]

$$\boldsymbol{P}(t+1) = \boldsymbol{P}(t) + \frac{\Delta}{K(t)}\boldsymbol{e}(t) - \frac{\Delta}{r - K(t)}[\boldsymbol{u} - \boldsymbol{e}(t)], \tag{10}$$

where $K(t)$ represents the number of actions to pursuit at time t and

$$e_j(t) = \begin{cases} 0, & \text{if} \quad \hat{d}_j(t) \leq \hat{d}_i(t) \cap j \neq i \\ 1 & \text{if} \quad \hat{d}_j(t) > \hat{d}_i(t) \cap j \neq i \end{cases}$$

$$e_i(t) = \begin{cases} 1, & \text{if} \quad \hat{d}_i(t) = \max\left\{\hat{d}_j(t)\right\} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Also in (10), Δ is referred to as the step size of the algorithm and results in a trade-off between convergence time (number of iterations) and convergence accuracy, that is, the probability that the algorithm converge to the optimal action. Larger Δ results in shorter convergence time and lower accuracy and vice versa. The algorithm is said to have converged to action $\alpha_c \in \mathcal{A}$ if the corresponding mass probability is grater than a probability convergence threshold $p_c \geq p_{Th}$.

B. Conditional-Inaction DGPA

Note in (10) that the actions probability update is conditioned solely on the current reward probability estimates $\hat{\boldsymbol{d}}(t)$. An action α_i may be rewarded even if its corresponding mass probability $p_i(t)$ reached zero as a result of a series of penalties being assigned to it. Therefore, discarded actions may be reconsidered during the learning process of the LA. To shorten the convergence time, we introduce the concept of *Conditional Inaction*, and propose a novel probability update policy. Based both on $\hat{\boldsymbol{d}}(t)$ and $\hat{\boldsymbol{P}}(t)$ the search space $\mathcal{A}$ is partitioned into three non-overlapping sets; indicating with $K(t)$ the number of actions to reward and $K'(t)$ those to penalize, we calculate the vector of increments $\boldsymbol{e}^I(t)$ as

$$e_j^I(t) = \begin{cases} 1, & \text{if} \quad \hat{d}_j(t) > \hat{d}_i(t) \quad \cup \quad p_j(t) \neq 0 \\ 0, & \text{if} \quad \hat{d}_j(t) \leq \hat{d}_i(t) \quad \cup \quad p_j(t) \neq 0 \\ 0, & \text{if} \quad p_j(t) = 0 \end{cases}$$

$$e_i^I(t) = \begin{cases} 1, & \text{if} \quad \hat{d}_i(t) = \max\left\{\hat{\mathbf{d}}(t)\right\} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

and the vector of decrements $\boldsymbol{e}^D(t)$ as

$$e_j^D(t) = \begin{cases} 0, & \text{if} \quad \hat{d}_j(t) > \hat{d}_i(t) \quad \cup \quad p_j(t) \neq 0 \\ 1, & \text{if} \quad \hat{d}_j(t) \leq \hat{d}_i(t) \quad \cup \quad p_j(t) \neq 0 \\ 0, & \text{if} \quad p_j(t) = 0 \end{cases}$$

$$e_i^D(t) = \begin{cases} 1, & \text{if} \quad \hat{d}_i(t) \neq \max\left\{\hat{\mathbf{d}}(t)\right\} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

and update the actions probability vector as follows.

$$\boldsymbol{P}(t+1) = \boldsymbol{P}(t) + \frac{\Delta}{K(t)}\boldsymbol{e}^I(t) - \frac{\Delta}{K'(t)}\boldsymbol{e}^D(t). \tag{14}$$

IV. Distributed Caching Based on LA

In this section, we propose a decentralized approach to the caching problem formulated in sec. II-A through a set of independent Learning Automata (LA). Let $\mathcal{D} = \{\delta_1, ..., \delta_D\}$ denote the set of independent Learners, E the *environment* which is the system the LAs learn from, and β a Reward Function (RF) or *feedback*. Given the CI-DGPA algorithm described in III-B, our goal is to design the tuple $\langle D_\mathrm{o}, E_\mathrm{o}, \beta_\mathrm{o}\rangle$ so that

$$\langle D_\mathrm{o}, E_\mathrm{o}, \beta_\mathrm{o}\rangle = \underset{\langle D,E,\beta\rangle}{\arg\min}\left\{\left|\bar{\Omega}_{\mathbf{A}_\mathrm{c}} - \bar{\Omega}_{\mathbf{A}_\mathrm{o}}\right|\right\}, \tag{15}$$

where $\bar{\Omega}_{\mathbf{A}}$ is the network average weighted delay corresponding to an allocation matrix $\mathbf{A}$.

A. Environment

The set of UTs scattered across the cell constitute the environment from which the system learns. Let N UTs be simultaneously active in the cell. At time $t = 0$, H Helpers broadcast their intention to cache M files and are informed from the MBS of the popularity of a library $\mathcal{F}$ of files. After a detection phase, each UT estimates the latency of d_{u_n} links with the respective providers Π_j, $j = 1, \cdots, H+1$. Each step of the learning process occurs over the following three consecutive time slots. (1) A *Learners selection* slot S_σ of duration T_σ when Learners select an action and broadcast their choice across the cell thus building an allocation matrix $\mathbf{A}(t)$. (2) A *users feedback* slot S_ϵ (T_ϵ) during which UTs identify the set of HSP-selected providers and send their feedbacks

$\epsilon \in \Re$ to the Learners. (3) An *environment feedback* slot S_β (T_β) during which each learner determines a feedback $\beta \in [0,1]$ for the action taken. The duration of a single game iteration can therefore be written as $T_I = T_\sigma + T_\epsilon + T_\beta$. If I_G denotes the number of iterations required in the game to converge to an allocation matrix, the game convergence time can be expressed as $T_{\mathbf{A}_c} = I_G \cdot T_I$.

B. Learners and Actions Space

Each Helper's cache can be represented with a set of M memory slots each capable of storing one entire file of B (bits). A one-to-one correspondence between learning agents and memory slots is assumed. Let $\mathbf{D}$ be the $M \times H$ matrix representing the set of Learners, where $\delta_{k,j}$ with $k = 1, \cdots, M$ and $j = 1, \cdots, H$, denotes the LA learning the k-th file in the cache of the j-th Helper. Learners (memory slots) take actions by selecting which files to cache from a set $\mathcal{A}_{k,j}$ of files where $\mathcal{A}_{k,j} \in \mathcal{F}$. Let $\mathbf{P}_{k,j}$, $\sigma_{k,j} \in \mathcal{A}_{k,j}$ and $\beta_{k,j}$ be respectively the actions probability vector, the currently selected file, and the corresponding feedback from the environment for learner $\delta_{k,j}$. We propose three strategies to determine the actions set $\mathcal{A}_{k,j}$ of each learner, also referred to as *learning schemes*; a *non cooperative* (NC), a *cooperative hard* (CH), and *cooperative shuffle* (CS) scheme. The first is a direct application of the learning algorithm described in sec. III-B where each learner operates on the totality of files in the library $\mathcal{A}_{k,j} = \{f_1, \cdots, f_F\}$. Because there is no correlation between the probability mass functions of any two Learners in the cell, redundancies might occur along the columns of the resulting allocation matrix $\mathbf{A}(t)$. This can be easily avoided by carefully designing the environment feedback, so that if two Learners of a same Helper take the same action, $\sigma_{k,j} = \sigma_{k',j}$ with $k < k'$, only learner $\delta_{k,j}$ can be rewarded by a UT. The second scheme is motivated by the convergence results of a single CI-DGPA over a random environment V. In CH the library $\mathcal{F}$ is partitioned into M non-overlapping subsets of H files so that

$$\mathcal{A}_{k,j} = \left\{ f_{[(k-1)\cdot H]+1}, \ \cdots, \ f_{(k\cdot H)} \right\}. \tag{16}$$

With smaller actions sets, this approach can substantially reduce the convergence time of each learner. Moreover, given the orthogonality of the actions sets $\mathcal{A}_{k,j}$ file duplications are inherently avoided. However, the files' position in the library, that is, the order of appearance of files in $\mathcal{F}$, is likely to compromise the performance of the algorithm. For instance, if files in $\mathcal{F}$ were placed in descending order according to their popularity, allocation matrices with two or more most popular files would be excluded from the space of possible solutions. In order to mitigate this effect while retaining the benefits of smaller actions sets, we propose a cooperative shuffle scheme. This motivated the third scheme (CS) where each Helper builds its set of actions independently, by uniformly selecting at random H files from the library $\mathcal{F}$. We refer to sec. V for a comparison of their performance.

C. Feedback

The N user terminals scattered across the cell constitute the environment from which the system learns. Let $\varepsilon_{k,j}^{(n)} \in \Re$, represent the feedback that user u_n sends to learner $\delta_{k,j}$ during S_ϵ. The design of $\varepsilon_{k,j}^{(n)}$ is based on the following considerations; more popular files account for a higher share of the users weighted delay as shown in Eq. (4); as a consequence of the HSP files selection policy, feedbacks should be distance-dependant, so that more weight is given to feedbacks from closer UTs. We correspond $\varepsilon_{k,j}^{(n)} > 0$ to positive feedbacks and $\varepsilon_{k,j}^{(n)} < 0$ to negative feedbacks. Moreover, we consider a symmetric scheme so that the feedback from user u_n over file f$_i$ can be written as

$$\varepsilon_{k,j}^{(n)} = \begin{cases} \Psi_\mathrm{R} = +l_i/\Gamma_n^{(i)} & \text{reward} \\ \Psi_\mathrm{P} = -l_i/\tau_{n,j} & \text{penalty} \end{cases}, \tag{17}$$

where l_i is the files's popularity and $\Gamma_n^{(i)}$ is the latency of the down-link with the HSP-selected j-th provider (3). For each learning agent the environment feedback $\beta_{k,j} \in [0,1]$ is determined based on the set $\{\varepsilon_{k,j}^{(n)}\}$ of users feedbacks received in a democratic proportional fashion. A cumulative users feedback $E_{k,j}$ is calculated at the Learners side

$$\mathrm{E}_{k,j} = \sum_{n=1}^{d_{h_j}} \varepsilon_{k,j}^{(n)}, \tag{18}$$

and an action is rewarded if and only if $\mathrm{E}_{k,j} > 0$, that is, a positive feedback is received from the majority of UTs. We propose two RFs referred to as *Weighted-Delay Based* WDB and *Average Weighted-Delay Based* AWDB.

D. WDB Feedback

The central idea behind the Weighted Delay Based (WDB) feedback is to achieve the objective function Eq. (6) through the minimization of the weighted delays of each UT. The feedbacks from UTs are both based on the set of HSP-selected provider $\tilde{\Pi}_j$ and the user's current Ω_n, i.e.,

$$\varepsilon_{k,j}^{(n)} = \begin{cases} \Psi_\mathrm{R} & \text{if} \quad \delta_{k,j} \in \tilde{\Pi}_j \cap \Omega_n \leq \Omega_{\min} \\ \Psi_\mathrm{P} & \text{if} \quad (\delta_{k,j} \in \tilde{\Pi}_j \cap \Omega_n > \Omega_{\min}) \cup \delta_{k,j} \notin \tilde{\Pi}_j \end{cases}, \tag{19}$$

where $\Omega_{\min}$ denotes the learner's current minimum weighted delay. The resulting environment feedback is positive ($\beta_{k,j} = 1$) if $\mathrm{E}_{k,j} > 0$, and negative ($\beta_{k,j} = 0$) otherwise.

E. AWDB Feedback

The AWDB reward function aims at rewarding a Learner only if its choice contributes to lower the AWD of the UTs connected to it. During the users feedback time-slot S_ϵ, UTs broadcast both their current weighted delays as from (4) and their individual feedback calculated as in WDB Sec. 4. The environment feedback $\beta_{k,j}$ is obtained as follows.

$$\beta_{k,j} = \begin{cases} 1 & \text{if} \quad \mathrm{E}_{k,j} > 0 \cap \bar{\Omega}^\mathrm{j} \leq \bar{\Omega}_{\min} \\ 0 & \quad\quad \text{otherwise} \end{cases}, \tag{20}$$

where $\bar{\Omega}^\mathrm{j}$ represents the average weighted delay of the set of users $\mathcal{D}_{h_j} = \{n : \tau_{n,h_j} < \infty\}$ connected to learner $\delta_{k,j}$.

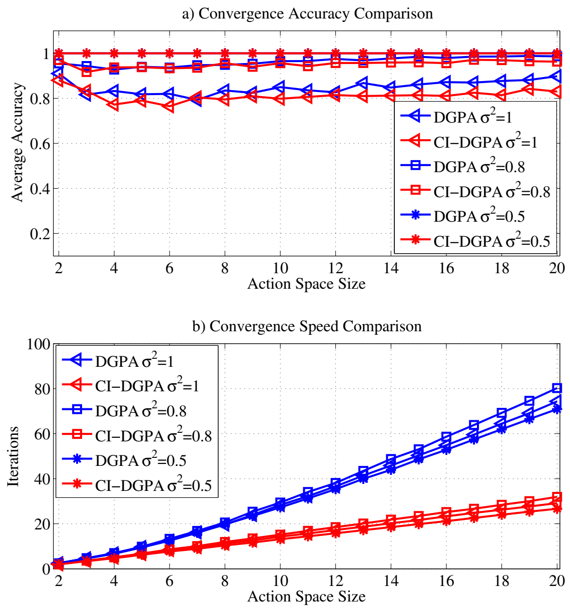

Fig. 2. Average accuracy and convergence speed (number of iterations) for CI-DGPA and DGPA over the Random Environment for different $\sigma^2_{\mathcal{A}}$.

F. Game Initialisation

The vector of estimates for the reward probabilities $\hat{\boldsymbol{d}}_{k,j}(0)$ is initially set to zero for any Learner in the game and is populated during the initialisation phase, referred to as *game initialisation*. Learners select files uniformly at random until each file is selected a minimum number of times denoted by $Z_{\min}$. At each iteration, the elements of $\hat{\boldsymbol{d}}_{k,j}$ are updated with the maximum likelihood estimation (MSE) of the reward probability $\hat{d}_i(t) = W_{f_i}(t)/Z_{f_i}(t)$, where $Z_{f_i}(t)$ and $W_{f_i}(t)$ denote respectively the number of times file f_i has been selected and rewarded at time t. The initialisation is assumed to be successful if $\widehat{\mathbf{d}}(0) \neq 0, \forall k \leq M, \forall j \leq H$.

V. Performance Evaluation

We evaluate accuracy and convergence speed of the proposed CI-DGPA in a Normal environment defined as follows. Let $\alpha_o \in \mathcal{A}$ be the optimal action in the set $\mathcal{A}$. The action taken by the LA is assessed against a current optimum $\hat{\alpha}(t) \in \mathcal{A}$, modeled by a discrete additive gaussian-like random process of power $\sigma^2_{\mathcal{A}}$ and expectation $E[\hat{\alpha}(t)] = \alpha_o$. Although impractical for many applications, a Normal environment can provide a lower bound on the performance of the LA, as well as a useful indication of its ability to cope with random fluctuations. Simulations over a maximum search space size of 20 actions show that CI-DGPA successfully meets a trade-off between convergence time and learning accuracy. As shown in Fig. 2, the average convergence time of CI-DGPA outperforms that of the DGPA and appears to increase linearly with the search space size. In terms of accuracy, CI-DGPA is comparable for $\sigma^2_{\mathcal{A}} < 1$, while a small gap can be observed for $\sigma^2_{\mathcal{A}} = 1$ and higher. Results were average by varying α_o across the entire search space, and repeating the learning process 100 times for each value of α_o. For an action set of size 20 and $\sigma^2_{\mathcal{A}} = 0.5$ both algorithms show a 100% accuracy with CI-DGPA considerably saving on number of iterations.

Note that in the multi-agent game of LA proposed in Sec. IV, information is exchanged between learners and UTs at each iteration. As a result, longer convergence times can reduce the spectral efficiency due to increased transmission overhead. Therefore, despite its slightly lower accuracy, the faster convergence of CI-DGPA is a promising feature for the feasibility of the decentralized caching algorithm.

A. Benchmark: The Greedy Placement

A common benchmark in wireless caching is represented by greedy algorithms. By definition, a greedy algorithm makes a locally optimal choice in the hope that this will lead to a globally optimal solution [10]. The process involves one Helper at a time, randomly selected from the set of Helpers $\mathcal{H}$. For each Helper, the MBS identifies the set of M files which minimizes the average weighted delay (AWD) of its d_{hj} surrounding UTs. Under the assumption of full channel state information (CSI), the MBS determines the UTs' files selection according to the HSP policy, and estimates the weighted delay of the UTs. The group of M files which result in the lowest AWD is cached in the current Helper. Note that no file duplicates are allowed in the cache of any Helper. Moreover, the files' position in the Helper's cache has no impact on the system performance. The Greedy algorithm searches among $\Upsilon_G = H \cdot \binom{F}{M}$ allocations, where $\binom{F}{M}$ is the Newton binomial, F is the size of the library $\mathcal{F}$ and M is the cache size of each Helper. The files popularity was modelled with a Zip-f distribution with discounted rate γ, where $0 \leq \gamma \leq 1$.

B. Numeric Results

Fig. 3 shows the network average weighted delay (NAWD) for different cache placements, as a function of the discounted rate γ. Let delay gain (DG) be the difference between the UTs average latency with and without caching. A first comparison can be drawn for the three learning schemes proposed in Sec. IV-B. Both NC and CS learning schemes achieve a comparable DG across the entire range of γs with the CS scheme converging twice as fast as the NC case (387.4 vs 808.0 respectively). Finally, the CH learning scheme exhibits a higher network delay (8.5% higher compared to CS or NC) and a higher convergence time (about 14% longer than CS) as a result of the hard partitioning of the search space $\mathcal{A}$. The same figure compares the NAWD resulting from the fastest learning scheme (CS) with the AWDB reward function (see Sec. IV-E), with that resulting from the greedy placement. For $\gamma = 0.7$, the proposed scheme shows a DG of 35.6%, while the greedy placement gains a 42.45%. The random placement completes Fig. 3 showing that the NAWD can be reduced of one fifth (21.3%) by populating the Helper's cache uniformly at random. In order to perform a statistical analysis of the latency experienced by the UTs, a set of 10^4 networks was generated. Fig. 4 compares the Cumulative Distribution Functions (CDFs) of the average delay (NAWD) resulting

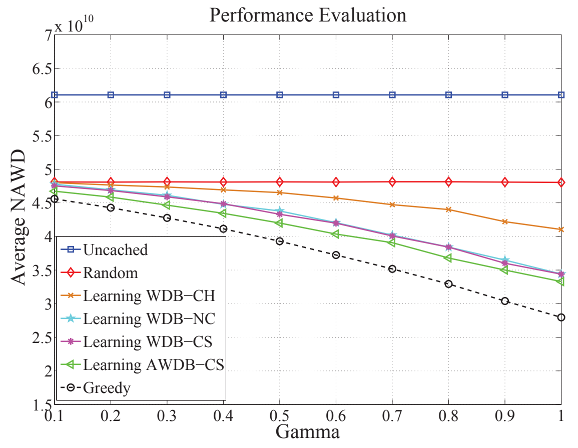

Fig. 3. NAWD Comparison for 10^4 networks with twelve Helper (H=12), M=3 and N=100 UTs randomly placed in a $1Km^2$ square cell.

TABLE I
DELAY GAIN AT THE 99-TH PERCENTILE OF THE NAWD

γ	$\Delta_{\bar{\Omega}_{0.01,G}}$ (%)	$\Delta_{\bar{\Omega}_{0.01,L}}$ (%)	$\Delta_{G,L}$
1	53.99	44.51	9.48
0.5	35.41	30.70	4.70
0.1	25.11	22.87	2.22

by the decentralized learning scheme based on CI-DGPA, with the one of a centralized greedy algorithm. Results are compared to the un-cached case. Overall, higher γs result in a larger performance gap between CI-DGPA and greedy. We denote with $\Delta_{\bar{\Omega}_{0.01},G}$ the difference in the 99-th percentile of the NAWD, between the un-cached network and the cache placement based on the greedy algorithm. Let $\Delta_{\bar{\Omega}_{0.01},L}$ be the corresponding difference for the CI-DGPA learning scheme and $\Delta_{G,L} = \Delta_{\bar{\Omega}_{0.01},G} - \Delta_{\bar{\Omega}_{0.01},L}$. Table I shows that $\Delta_{G,L}$ increases with γ. For $\gamma = 0.1$ the 99-th percentile of the NAWD with the proposed algorithm is only a 2% higher than the corresponding delay with a greedy centralized placement. The same value is reported to be a 4.70% and a 9.48% higher for $\gamma = 0.5$ and $\gamma = 1$ respectively. Results were simulated for a wireless network with $H = 12$, $M = 3$, and $N = 100$ with a team of 36 independent CI-DGPA LA as per Sec. III-B. Every channel in the network was assumed to be i.i.d AWGN with normalized noise power $\sigma^2 = 1$, transmission power P_{T_x} and identical path loss exponent $\alpha = 3$. Simulations are obtained for a 1Km^2 square cell, files of identical size $B = 10^7$ (bits), $\Delta = 1$, $Z_{\text{min}} = 5$ and a convergence probability threshold of 0.999. The learning process is considered to have converged when the last Learner reached convergence.

VI. Conclusions

In this paper a decentralized algorithm has been proposed to solve the cache placement problem in wireless networks, based on a game of independent Learning Automata (LA). Motivated by the need of reducing the signaling overhead, we have proposed a faster-converging DGPA algorithm for LA based on the concept of conditional inaction (CI-DGPA). We have designed two reward functions and empirically shown how the game convergence speed can be doubled by randomly sharing the actions space set among co-located learning agents. Statistical investigations have shown that the decentralized solution approaches the performance of a centralized greedy algorithm, ensuring lower computation complexity and bypassing the need for full CSI. Although a full cost-complexity analysis is yet to be developed, the distributed nature of CI-DGPA overcomes the need of communication channels among Helpers. This in turns reduces the overhead due to signaling and suggests the algorithm's suitability for real world implementation.

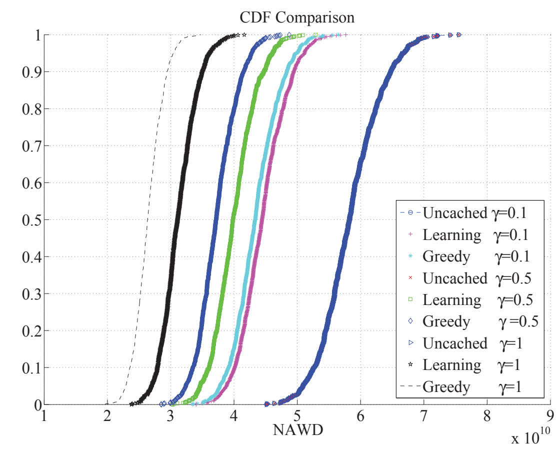

Fig. 4. Cumulative Distribution Functions (CDFs) of the NAWD for 10^4 networks with twelve Helper (H=12), M=3 and N=100 UTs randomly placed in a $1Km^2$ square cell.

References

[1] K. Shanmugam, N. Golrezaei, A. Dimakis, A. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.
[2] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11 520862.html.
[3] N. Golrezaei, A. Molisch, A. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Commun. Mag.*, vol. 51, no. 4, pp. 142–149, Apr. 2013.
[4] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.
[5] Y. Li, M. Qian, D. Jin, P. Hui, Z. Wang, and S. Chen, "Multiple mobile data offloading through disruption tolerant networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 7, pp. 1579–1596, July 2014.
[6] J. Gu, W. Wang, A. Huang, H. Shan, and Z. Zhang, "Distributed cache replacement for caching-enable base stations in cellular networks," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 2648–2653.
[7] P. Blasco and D. Gunduz, "Learning-based optimization of cache content in a small cell base station," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 1897–1903.
[8] M. Agache and B. Oommen, "Generalized pursuit learning schemes: new families of continuous and discretized learning automata," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 32, no. 6, pp. 738–749, Dec 2002.
[9] G. Horn and B. Oommen, "An application of a game of discrete generalised pursuit automata to solve a multi-constraint partitioning problem," in *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, vol. 2, Oct 2006, pp. 1042–1049.
[10] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein, "Introduction to algorithms," Third Edition.