Decentralized Learning for Multi-player Multi-armed Bandits

Dileep Kalathil, Naumaan Nayyar and Rahul Jain

Abstract—We consider the problem of distributed online learning with multiple players in multi-armed bandit models. Each player can pick among multiple arms. As a player picks an arm, it gets a reward from an unknown distribution with an unknown mean. The arms give different rewards to different players. If two players pick the same arm, there is a “collision”, and neither of them get any reward. There is no dedicated control channel for coordination or communication among the players. Any other communication between the users is costly and will add to the regret. We propose an online index-based learning policy called dUCB4, algorithm that trades off exploration v. exploitation in the right way, and achieves expected regret that grows at most near-$O(\log^2 T)$. The motivation comes from opportunistic spectrum access by multiple secondary users in cognitive radio networks wherein they must pick among various wireless channels that look different to different users.

Index Terms—Distributed adaptive control, multi-armed bandits, online learning, multi-agent systems.

I. INTRODUCTION

In [1], Lai and Robbins introduced the classical non-Bayesian multi-armed bandit model. Such models capture the essence of the learning problem that players face in an unknown environment, where the players must not only explore to learn but also exploit in choosing the best arm. Specifically, suppose a player can choose between $N$ arms. Upon choosing an arm $i$, it gets a reward from a distribution with density $f(x, \theta_i)$. Time is slotted, and players do not know the distributions (nor any statistics about them). The problem is to find a learning policy that minimizes the expected regret over some time horizon $T$. It was shown by Lai and Robbins that there exists an index-type policy that achieves expected regret that grows asymptotically as $\log T$, and this is order-optimal, i.e., there exists no causal policy that can do better. Anantharam, et al presented an extension to the case when the rewards are not i.i.d. but Markovian [2]. Since the mid-80s, many extensions and re-interpretations of Lai and Robbins index policy have been given, including by Agrawal [3] who presented a considerable simplification, and by Auer, et al [4], who presented many variations.

Recently, there is increasing interest in multi-armed bandit models, partly because of opportunistic spectrum access problems. Consider a user who must choose between $N$ wireless channels. Yet, it knows nothing about the channel statistics, i.e., has no idea of how good the channel it, and what rate it may expect to get on a particular channel. These have been formulated as multi-armed bandit problems, and index-type policies have been proposed for choosing spectrum channels. In many scenarios, there are multiple users accessing the channels at the same time. Each of these users must be matched to a different channel. These have been formulated as a combinatorial multi-armed bandit problem [5], and it was shown that an “index-matching” algorithm that at each instant determines a matching by solving a sum-index maximization problem achieves $O(\log T)$ regret, and this is indeed order-optimal.

In other settings, the users cannot coordinate, and the problem must be solved in a decentralized manner. Thus, settings where all channels (arms) are identical for all users have been considered, and index-type policies that can achieve coordination have been proposed that get $O(\log T)$ regret [6], [7]. The regret scales only polynomially in the number of users and channels.

In this paper, we consider the decentralized multi-armed bandit problem with distinct arms for each players. All players together must discover the best arms to play as a team. However, since they are all trying to learn at the same time, they may collide when two or more pick the same arm. We propose an index-type policy dUCB4 based on a variation of the UCB1 index. At its’ heart is a distributed bipartite matching algorithm such as Bertsekas’ auction algorithm [8]. We show that the dUCB4 algorithm that we introduce still achieves (at most) near-$O(\log^2 T)$ growth in expected regret.

II. MODEL AND PROBLEM FORMULATION

We consider an $N$-armed bandit with $M$ players. In a wireless cognitive radio setting, each arm could correspond to a channel, and each player to a user who wants to use a channel. Time is slotted, and at each
instant each player picks an arm. There is no dedicated control channel for coordination among the players. So, potentially more than one players can pick the same arm at the same instant. We will regard that as a collision. Player $i$ playing arm $k$ at time $t$ yields i.i.d. reward $S_{ik}(t)$ with univariate density function $f(s, \theta_{ik})$, where $\theta_{ik}$ is a parameter in the set $\Theta_{ik}$. We will assume that the rewards are bounded, and without loss of generality lie in $[0,1]$. Let $\mu_{i,k}$ denote the mean of $S_{ik}(t)$ w.r.t. the pdf $f(s, \theta_{ik})$. We assume that the parameter vector $\theta = (\theta_{ij}, 1 \leq i \leq M, 1 \leq j \leq N)$ is unknown to the players, i.e., the players have no information about the mean, the distributions or any other statistics about the rewards from various arms other than what they observe while playing. We also assume that each player can only observe the rewards that they get. When there is a collision, we will assume that all players that choose the arm on which there is a collision get reward zero. This can be relaxed to a setting where the players share the reward in some manner.

Let $X_{ij}(t)$ be the reward that player $i$ gets from arm $j$. Thus, if player $i$ plays arm $k$ at time $t$ (and there is no collision), $X_{ik}(t) = S_{ik}(t)$, and $X_{ij}(t) = 0, j \neq k$. Denote the action of player $i$ at time $t$ by $a_i(t) \in A := \{1, \ldots, N\}$. Then, the history seen by player $i$ at time $t$ is $H_i(t) = \{(a_i(1), X_{i,a_i(1)}(1)), \ldots, (a_i(t), X_{i,a_i(t)}(t))\}$ with $H_i(0) = \emptyset$. A policy $\pi_i = (\pi_i(t))_{t=1}^\infty$ for player $i$ is a sequence of maps $\pi_i(t): H_i(t) \rightarrow A$ that specifies the arm to be played at time $t$ given the history seen by the player. Let $\mathcal{P}(N)$ be the set of vectors such that

$$\mathcal{P}(N) := \{a = (a_1, \ldots, a_M): a_i \in A, a_i \neq a_j, \text{for } i \neq j\}.$$

The players have a team objective, namely over a time horizon $T$, they want to maximize the expected sum of rewards over some time horizon $T$, i.e., $\max \mathbb{E}\left[\sum_{t=1}^T \sum_{i=1}^M X_{i,a_i(t)}(t)\right]$. If the parameters $\mu_{i,j}$ are known, this could easily be achieved by picking a bipartite matching $k^* = \arg\max_{k \in \mathcal{P}(N)} \sum_{i=1}^M \mu_{i,k_i}$, i.e., the optimal bipartite matching with expected reward from each match. But the expected rewards are unknown. So, the players must pick learning policies that minimize the expected regret, defined for policies $\pi = (\pi_i, 1 \leq i \leq M)$ as

$$\mathcal{R}_\pi(T) = T \sum_{i} \mu_{i,k^*} - \mathbb{E}_\pi \left[ \sum_{i=1}^M \sum_{t=1}^T X_{i,\pi_i(t)}(t) \right].$$

Our goal is to find a decentralized algorithm that players can use such that together they minimize the regret.

III. SOME VARIATIONS ON SINGLE PLAYER MULTI-ARMED BANDITS

We first present some variations on the single player non-Bayesian multi-armed bandit model which will be used later for the multi-player problem.

A. UCB$_1$ with index recomputation every $L$ slots

Consider the classical single player non-Bayesian $N$-armed bandit problem. At each time $t$, the player picks a particular arm, say $j$, and gets a random reward $X_j(t)$. The reward $X_j(t), 1 \leq t \leq T$ are independent and identically distributed according to some unknown probability measure with an unknown expectation $\mu_j$. Without loss of generality, assume that $\mu_1 > \mu_2 > \cdots > \mu_N$, for $i = 2, \cdots, N$. Let $n_j(t)$ denote the number of times the arm $j$ has been played by time $t$. Denote $\Delta_j = \mu_1 - \mu_j$. Also, let $\Delta_{min} = \min_{j \neq \ell} \Delta_j$ and $\Delta_{max} = \max_j \Delta_j$. The regret for any policy $\pi$ is then given by

$$\mathcal{R}_\pi(T) := \mu_1 T - \sum_{j=1}^N \mu_j \mathbb{E}_\pi[n_j(T)].$$

Define an index, $g_j(t) = \overline{X}_j(t) + \sqrt{\frac{2 \log(t)}{n_j(t)}}$, (2)

where $\overline{X}_j(t)$ is the average reward obtained by playing arm $j$ by time $t$. It is defined as $\overline{X}_j(t) = \frac{\sum_{m=1}^t r_j(m)}{n_j(t)}$, where $r_j(t)$ is the reward obtained from arm $j$ at time $t$. If the arm $j$ is played at time $t$ then $r_j(m) = X_j(m)$ and otherwise $r_j(t) = 0$. Now, an index-based policy called UCB$_1$ [4] is to pick the arm that has the highest index at each instant. It can be shown that this algorithm achieves regret that grows logarithmically in $T$.

Asymptotic scaling was established by Lai and Robbins [1] and Agrawal [3], while non-asymptotic scaling was established by Auer, et al. [4].

An easy variation of the above algorithm which will be useful in our analysis of subsequence algorithms is the following. Suppose the index is re-computed only once every $L$ slots. In that case,

Theorem 1: Under the UCB$_1$ algorithm with recomputation of the index once every $L$ slots, the expected regret by time $T$ is given by

$$\mathcal{R}_{\text{UCB}_1}(T) \leq \sum_{j=1}^N \frac{8L \log T}{\Delta_j} + L \left(1 + \frac{\pi^2}{3}\right) \sum_{j=1}^N \Delta_j.$$ (3)

The proof follows [4] and taking into account the fact that every time a suboptimal arm is selected, it is played for the next $L$ slots rather than just one time.
B. UCB4 Algorithm when index computation is costly

Often, learning algorithms pay a penalty or cost for computation. This is particularly the case when the algorithms must solve combinatorial optimization problems that are NP-hard. Such costs also arise in decentralized settings wherein algorithms pay a communication cost for coordination between the decentralized players. This is indeed the case, as we shall see later when we present an algorithm to solve the decentralized multi-armed bandit problem. Here, however, we will just consider an “abstract” communication or computation cost. The problem we formulate below can be solved with better regret bounds than what we present. At this time, though, we are unable to design algorithms with better regret bounds that also help in decentralization.

Consider a computation cost every time the index is recomputed. Let the cost be $C$ units. Let $m(t)$ denote the number of times the index is computed by time $t$. Then, under policy $\pi$ the expected regret is

$$R_\pi(T) := \mu_1 T - \sum_{j=1}^{N} \mu_j E[n_j(T)] + C E[m(T)].$$

It is easy to argue that the UCB1 algorithm will give a regret $\Omega(T)$ for this problem. We present the UCB4 algorithm which gives sub-linear regret.

Define the UCB4 index, $g_j(t) := X_j(t) + \sqrt{\frac{2 \log(t)}{n_j(t)}}.

We define an arm $j^*(t)$ to be the best arm if $j^*(t) \in \arg \max_{1 \leq i \leq N} g_i(t)$. Let $X_1, \ldots, X_n$ be random variables with common range such that $E[X_i | X_1, \ldots, X_{i-1}] = \mu_i$. Let $S_n = \sum_{i=1}^{n} X_i$. Then for all $a \geq 0,$

$$P[S_n \geq n\mu + a] \leq e^{-2a^2/n}, P[S_n \leq n\mu - a] \leq e^{-2a^2/n}.$$ 

**Theorem 2:** The expected regret for the single player multi-armed bandit problem with per computation cost $C$ using the UCB4 algorithm is given by

$$R_{UCB4}(T) \leq \left( \Delta_{\max} + C(1 + \log T) \right) \left( \sum_{j=1}^{N} \frac{16 \log T}{\Delta_j^2} + 2N \right).$$

Thus, $R_{UCB4}(T) = O(\log^2 T)$.

The proof is long and due to page limitations is omitted. For the complete proof, see [10].

**Remarks.** 1. It is easy to show that the lower bound for the single player MAB problem with computation costs is $\Omega(\log T).$ This can be achieved by the UCB2 algorithm [4]. To see this, note that the number of times the player selects a suboptimal arm when using UCB2 is $O(\log T).$ Since $E[n_j(T)] = O(\log T),$ we get $E[\sum_{j=1}^{N} n_j(T)] = O(\log T),$ and also $E[m_1(T)] = O(\log T).$ Now, since the epochs are not getting reset after every switch and are exponentially spaced, the number of updates that result in the optimal allocation, $m_1(T) \leq \log T.$ These together, then yield

$$R_{UCB2}(T) \leq \sum_{j=1}^{N} E[n_j(T)] \cdot \Delta_j + C E[m(T)] = O(\log T).$$

2. Variations of the UCB2 algorithm that use a deterministic schedule can also be used. But none of these algorithms can be used in solving the decentralized multi-armed bandit problem that we introduce in the next section. This is the main reason for introducing the UCB4 algorithm.

C. Algorithms with finite precision indices

Often, there might be a cost to compute the indices to a particular precision. In that case, indices may be known up to some $\epsilon$ precision, and it may not possible to tell which of two indices is greater if they are within $\epsilon$ of each other. The question then is how is the performance of various index-based policies such as UCB1, UCB4, etc. affected if there are limits on index resolution, and only an arm with an $\epsilon$-highest index can be picked. We first show that if $\Delta_{\min}$ is known, we can fix a precision $\epsilon < \Delta_{\min},$ so that UCB4 algorithm will achieve order log-squared regret growth with $T$. If $\Delta_{\min}$ is not known, we can pick a sequence $\{\epsilon_i\}$ such that $\epsilon_i \to 0,$ as $t \to \infty.$
Denote the cost of computation for $\epsilon$-precision be $C(\epsilon)$. We assume that $C(\epsilon) \to \infty$ monotonically as $\epsilon \to 0$.

Theorem 3: (i) If $\Delta_{\text{min}}$ is known, choose an $\epsilon < \Delta_{\text{min}}$. Then, the expected regret of the UCB4 algorithm with $\epsilon$-precise computations is given by
\[
\tilde{R}_{\text{UCB}4}(T) \leq (\Delta_{\text{max}} + C(\epsilon)(1 + \log T)) \cdot \left( \sum_{j > 1} \frac{16 \log T}{(\Delta_j - \epsilon)^2} + 2N \right).
\]

Thus, $\tilde{R}_{\text{UCB}4}(T) = O(\log^2 T)$.

(ii) If $\Delta_{\text{min}}$ is unknown, denote $\epsilon_{\text{min}} = \Delta_{\text{min}}/2$ and choose a sequence $\{\epsilon_t\}$ such that $\epsilon_t \to 0$ as $t \to \infty$. Then, there exists a $t_0 > 0$ such that for all $T > t_0$,
\[
\tilde{R}_{\text{UCB}4}(T) \leq (\Delta_{\text{max}} + C(\epsilon_{\text{min}})) t_0 + (\Delta_{\text{max}} + C(\epsilon_T)(1 + \log T)) \cdot \left( \sum_{j > 1} \frac{16 \log T}{(\Delta_j - \epsilon_{\text{min}})^2} + 2N \right)
\]

where $t_0$ is such that $\epsilon_{t_0} = \epsilon_{\text{min}}$. For $C(\epsilon) = \log 1/\epsilon$ and $\epsilon_t = 1/\log(k) t$ (log iterated $k$ times), we get $\tilde{R}_{\text{UCB}4}(T) = O(\log^{(k+1)} T \cdot \log^2 T)$.

The proof is omitted due to page limitations. For the complete proof, see [10].

Remarks. The regret can be made arbitrarily close to $O(\log^2 T)$ by choosing a sequence $\{\epsilon_t\}$ that decreases arbitrarily slowly to zero. This, will however cause $t_0$ to be very large.

IV. THE DECENTRALIZED MAB PROBLEM

We now consider the decentralized multi-armed bandit problem wherein multiple players are playing at the same time. Players have no information about means or distributions of rewards from various arms. There are no dedicated control channels for coordination or communication between the players. If two or more players pick the same arm, we assume that neither gets any reward. Thus, this is an online learning problem of distributed bipartite matching.

Distributed algorithms for bipartite matching algorithms are known [8] which determine an $\epsilon$-optimal matching with a ‘minimum’ amount of information exchange and computation. However, every run of this distributed bipartite matching algorithm incurs a cost, which is a combination of computation cost and communication cost to exchange information necessary for decentralization. Let $C$ be the cost per run, and $m(t)$ denotes the number of times the distributed bipartite matching algorithm is run by time $t$. Then, under policy $\pi$, the expected regret is given by
\[
\mathcal{R}_\pi(T) = T \sum_{i=1}^{M} \mu_i k_i^* - \mathbb{E}_\pi \left[ \sum_{i=1}^{T} \sum_{j=1}^{M} X_{i,j} (t) \right] + C E[m(t)].
\]

Temporal Structure. We will divide time into frames. Each frame is one of two kinds: a decision frame, and an exploitation frame. In the decision frame, the index will be recomputed, and the distributed bipartite matching algorithm run again to determine the new matching. The length of such a frame can be seen as cost of the algorithm. In the exploitation frame, the current matching is exploited without updating the indices. When a suboptimal matching is played in such frames, it contributes to regret. Later, we will allow length of the frames to increase with time.

We now present the dUCB4 algorithm, a decentralized version of UCB4. For each player $i$ and each arm $j$, we define a dUCB4 index at the end of frame $t$ as
\[
g_{i,j}(t) = \bar{X}_{i,j}(t) + \sqrt{\frac{(M+2) \log n_i(t)}{n_{i,j}(t)}} \quad (5)
\]

where $n_i(t)$ is the number of successful plays (without collisions) of player $i$ by frame $t$, $n_{i,j}(t)$ is the number of times player $i$ picks arm $j$ successfully by frame $t$. $\bar{X}_{i,j}(t)$ is the sample mean of rewards from arm $j$ for player $i$ from $n_{i,j}(t)$ samples. Let $g(t)$ denote the vector $(g_{i,j}(t), i = 1 : M; j = 1 : N)$. Note that $g$ is computed only in the decision frames using the information available up to that time. Each player now uses the dUCB4 algorithm. We will refer to an $\epsilon$-optimal distributed bipartite matching algorithm as $dBM_\epsilon(g(t))$ that yields a solution $k^*(t) := (k_1^*(t), \ldots, k_M^*(t)) \in \mathcal{P}(N)$ such that
\[
\sum_{i=1}^{M} g_{i,k_i^*(t)}(t) \geq \sum_{i=1}^{M} g_{i,k_i}(t) - \epsilon, \, \forall k \in \mathcal{P}(N), k \neq k^*.
\]

Let $k^* \in \mathcal{P}(N)$ be such that $k^* = \arg \max_{k \in \mathcal{P}(N)} \sum_{i=1}^{M} \mu_i k_i$, i.e., an optimal bipartite matching with expected rewards from each matching. Denote $\mu^* := \sum_{i=1}^{M} \mu_i k_i^*$, and define $\Delta_k := \mu^* - \sum_{i=1}^{M} \mu_i k_i$, $k \in \mathcal{P}(N)$. Let $\Delta_{\text{min}} = \min_{k \in \mathcal{P}(N), k \neq k^*} \Delta_k$ and $\Delta_{\text{max}} = \max_{k \in \mathcal{P}(N)} \Delta_k$. We assume that $\Delta_{\text{min}} > 0$.

In dUCB4, at the end of every decision frame, the $dBM_\epsilon(g(t))$ will give a legitimate matching, with no two players colliding on any arm. Thus, the regret accrues either when the matching $k(t)$ is not the optimal matching $k^*$, or when a decision frame is employed by the players to recompute the matching. Every time a frame is a decision frame, it adds a cost $C$ to the
Algorithm 2 dUCB4 for User i

1: Initialization: Play a set of matchings so that each player plays each arm at least once. Set counter $\eta = 1$.

2: while ($t \leq T$) do
3:   if ($\eta = 2^p, p = 0, 1, 2, \ldots$) then
4:     \text{//Decision frame:}
5:     Update $g(t)$;
6:     Participate in the dBM $(g(t))$ algorithm to obtain a match $k^*_i(t)$;
7:     if ($k^*_i(t) \neq k^*_i(t-1)$) then
8:       Use an INTERRUPT to notify all other players about changed allocation;
9:       Reset $\eta = 1$;
10:     end if
11:   else
12:     \text{// Exploitation frame:}
13:     $k^*_i(t) = k^*_i(t-1)$;
14:     end if
15:     Play arm $k^*_i(t)$;
16:     Increment counter $\eta = \eta + 1$, $t = t + 1$;
17: end while

Regret. The cost $C$ depends on two parameters: (a) the precision of the bipartite matching algorithm $\epsilon_1$, and (b) the precision of the index representation $\epsilon_2$. A bipartite matching algorithm has an $\epsilon_2$-precision if it gives an $\epsilon_1$-optimal matching. This would happen, for example, when such an algorithm is run only for a finite number of rounds. The index has an $\epsilon_2$-precision if any two indices are not distinguishable if they are closer than $\epsilon_2$. This can happen for example when indices must be communicated to other players in a finite number of bits.

Thus, the cost $C$ is a function of $\epsilon_1$ and $\epsilon_2$, and can be denoted as $C(\epsilon_1, \epsilon_2)$, with $C(\epsilon_1, \epsilon_2) \to \infty$ as $\epsilon_1$ or $\epsilon_2 \to 0$. Since, $\epsilon_1, \epsilon_2$ are the parameters that can be fixed a priori, we choose $\epsilon_1 = \epsilon_2 = \epsilon$ to simplify the algorithm description and analysis. We denote the cost as $C(\epsilon)$ and is assumed to be monotonically decreasing.

We first show that if $\Delta_{\text{min}}$ is known, we can choose an $\epsilon < \Delta_{\text{min}}/(M + 1)$, so that dUCB4 algorithm will achieve order log-squared regret growth with $T$. If $\Delta_{\text{min}}$ is not known, we can pick a sequence $\{\epsilon_t\}$ such that $\epsilon_t \to 0$, as $t \to \infty$. In a decentralized bipartite matching algorithm, the precision $\epsilon$ will depend on the amount of information exchanged in the decision frames. It, thus, is some monotonically decreasing function $\epsilon = f(L)$ of their length $L$ such that $\epsilon \to 0$ as $L \to \infty$. Thus, we must pick a sequence $\{L_t\}$ such that $L_t \to \infty$. Clearly, $C(f(L_t)) \to \infty$ as $t \to \infty$. This can happen arbitrarily slowly.

Theorem 4: (i) Let $L$ be the length of a frame. If $\Delta_{\text{min}}$ is known, choose $L = L_{\text{min}}$ such that $f(L_{\text{min}}) < \Delta_{\text{min}}/(M + 1)$. Then, the expected regret of the dUCB4 algorithm is

$$\tilde{R}_{\text{dUCB4}}(T) \leq (L_{\text{min}} \Delta_{\text{max}} + C(f(L_{\text{min}})))(1 + \log T) \left(4M^2(M + 2)N \log T + \frac{(4M^3(M + 2)N \log T + NM(2M + 1))}{(\Delta_{\text{min}} - \epsilon_{\text{min}})^2}ight).$$

Thus, $\tilde{R}_{\text{dUCB4}}(T) = O(\log^2 T)$.

(ii) When $\Delta_{\text{min}}$ is unknown, denote $\epsilon_{\text{min}} = \Delta_{\text{min}}/(2(M + 1))$ and let $L_t \to \infty$ as $t \to \infty$. Then, there exists a $t_0 > 0$ such that for all $T > t_0$,

$$\tilde{R}_{\text{dUCB4}}(T) \leq (L_t \Delta_{\text{max}} + C(f(L_t)))(1 + \log T) \left(4M^3(M + 2)N \log T + NM(2M + 1)\right),$$

where $t_0$ is such that $\epsilon_{\text{min}} = f(L_{t_0})$.

For $\epsilon_t = 2^{-L_t}$, $C(\epsilon_t) = \log 1/\epsilon_t$, and $L_t = \log \log t$, we get $\tilde{R}_{\text{dUCB4}}(T) = O(\log \log T \cdot \log^2 T)$.

The proof is omitted due to page limitations. For the complete proof, see [10].

Remark. The UCB2 algorithm described in [4] performs computations only at exponentially spaced time epochs. So, it is natural to imagine that a decentralized algorithm based on it could be developed, and get a better regret bound. Unfortunately, the single player UCB2 algorithm has an obvious weakness: regret is linear in the number of arms. Thus, the decentralized/combinatorial extension of UCB2 would yield regret growing exponentially in the number of players and arms.

V. DISTRIBUTED BIPARTITE MATCHING: ALGORITHM AND IMPLEMENTATION

We now present one such algorithm, namely, Bertsekas’ auction algorithm [8], and its distributed implementation. We note that the presented algorithm is not the only one that can be used. The dUCB4 algorithm will work with a distributed implementation of any bipartite matching algorithm.

Consider a bipartite graph with $M$ players on one side, and $N$ arms on the other, and $M \leq N$. Each player has a value $\mu_{i,j}$ for each arm $j$. Each player knows only his own values.

Let us denote by $k^*_j$, a matching that maximizes the matching surplus $\sum_{i,j} \mu_{i,j} x_{i,j}$, where the variable $x_{i,j}$
is 1 if \( i \) is matched with \( j \), and 0 otherwise. Note that \( \sum_j x_{i,j} \leq 1, \forall j \), and \( \sum_i x_{i,j} \leq 1, \forall i \). Our goal is to find an \( \epsilon \)-optimal matching. We call any matching \( k^* \) to be \( \epsilon \)-optimal if \( \sum_i \mu_{i,k^*(i)} - \sum_i \mu_{i,k^*(i)} \leq \epsilon \). Here, \( 2 \max_j \) is the second highest maximum over all \( j \). The best arm for a player \( i \) is arm \( j_i^* = \arg \max_j (\mu_{i,j} - p_j) \).

The winner \( i_j^* \) on an arm \( j \) is the player who submitted the highest bid on that arm.

**Algorithm 3 : dBM** (Bertsekas Auction Algorithm)

1. All players \( i \) initialize prices \( p_j = 0, \forall \) channels \( j \);
2. while (prices change) do
3. Player \( i \) communicates his preferred arm \( j_i^* \) and bid \( b_i = \max_j (\mu_{i,j} - p_j) - 2 \max_j (\mu_{i,j} - p_j) + \frac{1}{M} \) to all other players.
4. Each player determines on his own if he is the winner \( i_j^* \) on arm \( j \);
5. All players set prices \( p_j = \mu_{i_j^*, j} \);
6. end while

The following lemma in [8] establishes that Bertsekas’ auction algorithm will find the \( \epsilon \)-optimal matching in a finite number of steps, with an upper bound that depends on problem primitives.

**Lemma 1:** Given \( \epsilon > 0 \), Algorithm 3 with rewards \( \mu_{i,j} \), for player \( i \) playing the \( j \)th arm, converges to a matching \( k^* \) such that \( \sum_i \mu_{i,k^*(i)} - \sum_i \mu_{i,k^*(i)} \leq \epsilon \) where \( k^* \) is an optimal matching. Furthermore, this convergence occurs in less than \( M \log \max_j (\mu_{i,j}) \) iterations.

**The temporal structure:** Time is divided into frames of length \( L \). Each frame is either a decision frame, or an exploitation frame. In the exploitation frame, each player plays the arm it was allocated in the last decision frame. The distributed bipartite matching algorithm (Algorithm 3), is run in the decision frame. The decision frame has an interrupt phase of length \( M \) and negotiation phase of length \( L - M \).

**Interrupt Phase:** The interrupt phase can be implemented very easily. It has length \( M \) time slots. On a pre-determined channel, each player by turn transmits a ‘\( 1 \)’ if the arm with which it is now matched has changed, ‘\( 0 \)’ otherwise. If any user transmits a ‘\( 1 \)’, everyone knows that the matching has changed, and they reset their counter \( \eta = 1 \).

**Negotiation Phase:** The information needed to be exchanged to compute an \( \epsilon \)-optimal matching is done in the negotiation phase. We first provide a packetized implementation of the negotiation phase. The negotiation phase consists of \( J \) subframes of length \( M \) each (see figure 1). In each subframe, the users transmit a packet by turn. The packet contains bid information: (channel number, bid value). Since all users transmit by turn, all the users know the bid values by the end of the subframe, and can compute the new allocation, and the prices independently. The length of the subframe, \( J \), determines the precision \( \epsilon \) of the distributed bipartite matching algorithm.

If a packetized implementation is not possible, we can give a physical implementation. Our only assumption here is going to be that each user can observe a channel, and determine if there was a successful transmission on it, a collision, or no transmission, in a given time slot. The whole negotiation phase is again divided into \( J \) sub-frames. In each sub-frame, each user transmits by turn. It simply transmits \( \lceil \log M \rceil \) bits to indicate a channel number, and then \( \lceil \log 1/\epsilon \rceil \) bits to indicate its bid value to precision \( \epsilon \). The number of such sub-frames \( J \) is again chosen so that the dBM algorithm (based on Algorithm 3) returns an \( \epsilon_2 \)-optimal matching.

**Fig. 1.** Figure showing structure of decision frame.

**REFERENCES**


