Online Learning for Active Cache Synchronization

Andrey Kolobov [1] **Sébastien Bubeck** [1] **Julian Zimmert** [2]

Abstract

Existing multi-armed bandit (MAB) models make two implicit assumptions: an arm generates a payoff only when it is played, and the agent observes every payoff that is generated. This paper introduces SYNCHRONIZATION BANDITS, a MAB variant where *all* arms generate costs at *all* times, but the agent observes an arm's instantaneous cost only when the arm is played. SYNCHRONIZATION MABs are inspired by online caching scenarios such as Web crawling, where an arm corresponds to a cached item and playing the arm means downloading its fresh copy from a server. We present MIRRORSYNC, an online learning algorithm for SYNCHRONIZATION BANDITS, establish an adversarial regret of $O(T^{2/3})$ for it, and show how to make it practical.

1. Introduction

Multi-armed bandits (MAB) (Robbins, 1952) have been widely applied in settings where an agent repeatedly faces K choices (*arms*), each associated with its own payoff distribution unknown to the agent at the start, and needs to eventually identify the arm with the highest mean payoff by pulling a subset of arms at a time and observing a payoff sampled from their distributions. MABs' defining property is that the agent observes an arm's instantaneous payoff when and only when the agent plays it. A key hidden assumption that goes hand-in-hand with it in the existing bandit models is that each arm generates reward when and only when it is played, which, combined with the bandit feedback property, also implies that the agent observes all generated payoffs.

In this paper, we go beyond these seemingly fundamental assumptions by identifying a class of practical settings that violate them and analyzing it using online learning theory. Specifically, this paper formalizes scenarios that we call SYNCHRONIZATION *MABs*. In these settings, the agent can be thought of as holding copies of K files whose originals come from different remote sources. As time goes by, the files change at the sources, and their copies increasingly differ from the originals, becoming *stale*. The agent's task is to refresh these files by occasionally downloading their new copies from remote sources, under a constraint B on the average number of downloads per time unit.

For each file, the agent is *continually* penalized for its staleness. The expected penalty at each time step due to this file is a non-decreasing function of the time since the file's last refresh. Playing arm k here corresponds to refreshing file k: doing so temporarily reduces its staleness and thereby diminishes the cost incurred due to it per time unit. The goal is to find a synchronization policy that minimizes regret in terms of the average staleness penalties by refreshing files according to a well-chosen schedule.

Crucially, at any moment the agent doesn't know how outdated its copy of a given file is, except at the time when it downloads its fresh copy, and therefore *most of the time doesn't know the penalties it is incurring*. It observes the penalty only when it plays an arm, i.e., refreshes a file, and has a chance to see how different the cached copy was right before the refresh. Even this action reveals only the *instantaneous* penalty due to this file, not the cumulative penalty the file has brought on since its last refresh.

SYNCHRONIZATION MABs are inspired by problems such as web crawl scheduling (Wolf et al., 2002; Cho & Garcia-Molina, 2003a; Azar et al., 2018; Kolobov et al., 2019a; Upadhyay et al., 2020) and database update management (Gal & Eckstein, 2001; Bright et al., 2006). All these settings involve a cache that must *proactively* initiate downloads to refresh its content. This is in contrast to, e.g., Web browser caches that *passively* monitor a stream of download requests initiated by another program. The few existing works on policy learning for active caching (Kolobov et al., 2019a; Upadhyay et al., 2020) apply only to specific penalty functions. In contrast, the theoretical results in this paper are independent of the penalties' functional form, and come with a practical online learning strategy for this model.

High-level analysis idea and paper outline. Online learning theory is a powerful tool for analyzing decision-making models where an agent operates in discrete-time instantaneous rounds by playing a candidate solution (arm), imme-

[1]Microsoft Research, Redmond [2]Google Research, Berlin. Work on this paper partially done during a visit to Microsoft Research, Redmond. Correspondence to: Andrey Kolobov <akolobov@microsoft.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

diately getting a feedback on it (a sample from the arm's payoff distribution), and using it to choose a candidate solution for the next round. Unfortunately, online learning's traditional assumptions clash with the properties of our setting. As Section 2 describes, SYNCHRONIZATION MAB is a *continuous-time* model with *non-stationary* sparsely observable costs. Its candidate solutions are multi-arm *policies* (Section 3). Getting useful feedback on a policy, such as an estimate of its cost function or gradient, isn't instantaneous; it requires playing the policy for a non-trivial stretch of time.

In Section 4, we present the MIRRORSYNC algorithm, which continuously plays a candidate policy along with "exploratory" arm pulls, periodically updating it with online mirror descent (Nemirovsky & Yudin, 1983; Bubeck, 2016). It uses a novel unbiased policy gradient estimator that operates in the face of sparse policy cost observations. Our regret analysis of MIRRORSYNC in Section 5 critically relies on the convexity of policy cost functions – a property we derive in Section 3 from minimal assumptions on SYNCHRONIZATION MABs' payoffs. The regret analysis treats time intervals between MIRRORSYNC's policy updates as learning "rounds" and thereby brings online learning theory to bear on SYNCHRONIZATION BANDITS. Section 6 introduces ASYNCMIRRORSYNC, a practical MIRRORSYNC variant that lifts MIRRORSYNC's idealizing assumptions. In Section 8, we compare the two algorithm empirically.

The contributions of this paper are thus as follows:

(1) We cast active caching as an online learning problem with sparse feedback, enabling principled theoretical analysis of this setting under a variety of payoff distributions.

(2) Based on this formulation, we propose a theoretic strategy for active caching under unknown payoff distributions and derive an adversarial regret bound of $O(T^{2/3})$ for it. In doing so, we overcome the challenges of sparse and temporal feedback inherent in this scenario that existing online learning theory does not address.

(3) We present a practical variant of the above strategy that lifts the latter's assumptions and, as experiments demonstrate, has the same empirical convergence rate.

2. Model formalization

SYNCHRONIZATION BANDITS are a *continuous-time* MAB model with K arms. Other than operating in continuous-time it differs from existing MAB formalisms in the mechanism by which arms generate costs/rewards and the observability of the generated costs from the agent's standpoint. In this section we detail both of these aspects, using the aforementioned cache update scenario as an illustrating example.

Cost-generating processes. In SYNCHRONIZATION MABs, every arm k incurs a stochastically generated cost $\hat{c}_{k,t}$ *at every time instant* t, whether the arm is played or not. However, the distribution of arm k's possible instantaneous costs at time t depends on how much time has passed since the last time arm k was played to refresh the corresponding cached item. We denote the length of this time interval as $\tau_k(t) \in [0, \infty)$. Thus, arm k's cost generation process is described by a family of random variables $\{c_k(\tau_k(t)) | t \geq 0\}$ as stated in the following assumption:

Assumption 1. *(Cost generation) At time t, each arm incurs a cost independently of being played by the agent. The instantaneous cost $\hat{c}_{k,t}$ due to arm k at time t is sampled from a random variable $c_{k,t} = c_k(\tau_k(t))$ s.t.: (1) $\tau_k(0) \triangleq 0$; (2) $c_k(0) = DiracDelta(0)$; (3) there exists a bound $U < \infty$ s.t. $supp(c_k(\tau)) \subseteq [0, U]$ for every arm's cost generation process c_k and any time interval length $\tau \geq 0$.*

By this assumption, for every arm and any amount of time τ since its latest play, its cost expectation is well-defined:

$$\overline{c}_k(\tau) \triangleq \mathbb{E}[c_k(\tau)]$$

Agent's knowledge and cost observability. While costs are generated by all arms continually, in our model the agent doesn't observe most of them, with an important exception:

Assumption 2. *(Cost knowledge and observability) For each arm k, the agent observes a cost $\hat{c}_{k,t} \sim c_{k,t}$ at time t if and only if the agent plays arm k at that time. The agent doesn't know the distributions of random variables $c_{k,t}$.*

Assumption 2 is crucial in two ways. First, it means that our model provides only bandit feedback. Namely, the agent doesn't see arms' costs at all times, unlike in related models such as maintenance scheduling (Bar-Noy et al., 1998). Second, coupled with Assumption 1 it implies that there is no causal relationship between playing an arm and incurring a cost, which is an implicit assumption that standard bandit strategies rely on.

Arm play modes. At any time t, any of SYNCHRONIZATION MAB's arms can be played in one of two modes:

Sync mode. Playing arm k in this mode at time t resets the arm's state, i.e., sets $\tau_k(t) \leftarrow\!\shortmid 0$. In addition, per Assumption 2, the agent observes the arm's instantaneous cost sample $\hat{c}_{k,t}$ immediately before $\tau_k(t)$ is reset to 0.

In the case of a cache, this means downloading a fresh copy of file k, estimating the difference between k's current original and the cached copy, and overwriting the cached copy with the new one.

Probe mode. By playing arm k allows in probe mode, the agent observes the arm's instantaneous cost, but the arm's state $\tau_k(t)$ is not reset.

In caching settings, this corresponds to downloading a fresh copy of item k, but using it purely to estimate the difference between k's current original and the cached copy, without overwriting the cached copy.

Since, by Assumption 1, $\overline{c}_k(0) = 0$, playing an arm in sync mode gives the agent a way to temporarily reduce the expected rate at which the arm incurs costs. However, due to the following assumption, after a sync play the arm's cost generation rate starts growing again:

Assumption 3. *(Cost monotonicity) For every arm k, the means $\overline{c}_k(\tau_k(t))$ of instantaneous cost random variables $c_k(\tau_k(t))$ are non-decreasing in time since the latest sync-mode play $\tau_k(t)$. If arm k was played in sync mode at time t_0, then any sequence of arm k's cost observations $\hat{c}_{k,1}, \hat{c}_{k,2}, \ldots$ yielded by probe-mode plays after t_0 and until this arm's next sync-mode play at time t_0' is non-decreasing.*

Arm state $\tau_k(t)$ can be viewed as the amount of time that has passed since the arm's last sync by time point t; the more time has passed, the more cost the arm is incurring per time unit. Playing an arm in sync mode simply resets this time counter. Thus, according to Assumption 3, not only does the *total* cost generated by arm k since its previous sync play grow as time goes by – which is to be expected – but so does the rate at which it happens.

Note that probe-mode arm plays don't help the agent reduce running costs directly. Instead, as we show in Section 4, they help the agent learn a good arm-playing policy faster.

Example. All of the above assumptions are natural in real-world scenarios that inspired the SYNCHRONIZATION model. For instance, in Web crawling each online web page accumulates changes according to a temporal process $\mathscr{D}_k(t)$, which is widely assumed to be Poisson (i.e., memoryless) in the Web crawling literature (Wolf et al., 2002; Cho & Ntoulas, 2002; Cho & Garcia-Molina, 2003a;b; Azar et al., 2018; Kolobov et al., 2019a;b; Upadhyay et al., 2020). For each indexed page, the agent (the search engine) incurs a cost $\mathscr{C}_k(d)$ due to serving outdated search results, as a function of the total difference d between the indexed page copy and the online original. From this perspective, $c_k(\tau) \triangleq \mathscr{C}_k(\mathscr{D}_k(\tau))$, but at least one other approach models $c_k(\tau)$ directly as a function of a web page copy's age (Cho & Garcia-Molina, 2000). In either case, Assumption 3 holds: the more time passes since the page's last crawl, the higher the expected instantaneous penalty. Moreover, penalties don't decrease between two successive crawls of a page: e.g., in case changes are generated by a Poisson process, their number can only grow with time since last refresh, and so can the penalty.

3. Policies and their cost functions

In order to derive a learning strategy for SYNCHRONIZATION BANDITS (Section 4) and its regret analysis (Section 5), we first derive the necessary building blocks: the cost of an arbitrary policy for this model, the class of policies that will serve as our algorithm's hypothesis space, and parameterized cost functions for policies of this class.

Policy costs. Our high-level aim is finding a SYNCHRONIZATION policy π that has a low expected average cost over an infinite time horizon. Whether a policy π is history-dependent, stochastic, or neither, executing it produces a *schedule* $\sigma_k = ((t_1, l_1), (t_2, l_2), \ldots)$ for each arm k, a possibly infinite sequence of time points t_{n_k} when the arm is to be played and corresponding labels l_{n_k} specifying whether the arm should be played in probe or sync mode at that time. For convenience, WLOG assume that t_0 always refers to $t = 0$, let $\tau_{n_k} \triangleq t_{n_k} - t_{n_k - 1}$, and for any finite horizon H let $N_k(H)$ be the index of schedule σ_k's largest time point not exceeding H:

$$N_k(H) \triangleq \begin{cases} argmax_{n \in \mathbb{N}}\{t_n \in \sigma_k | t_n \leq H\} \text{ if such } n \text{ exists} \\ \infty \text{ otherwise} \end{cases} \tag{1}$$

Given this definition, let $t_{N_k(H)+1} \triangleq H$.

Recalling that each arm has a specific time-dependent cost distribution $c_k(\tau_k(t))$ with mean $\overline{c}_k(\tau_k(t))$, we define the average infinite-horizon cost $J_k^{\sigma_k}$ of arm k's schedule σ_k as

$$\begin{aligned} J_k^{\sigma_k} &\triangleq \liminf_{H \to \infty} \mathbb{E}\left[\frac{1}{H} \sum_{n_k=1}^{N_k(H)+1} \int_0^{\tau_{n_k}} c_k(\tau) d\tau\right] \\ &= \liminf_{T \to \infty} \frac{1}{H} \sum_{n_k=1}^{N_k(H)+1} \int_0^{\tau_{n_k}} \overline{c}_k(\tau) d\tau \end{aligned} \tag{2}$$

Letting

$$\overline{C}_k(\tau') \triangleq \int_0^{\tau'} \overline{c}_k(\tau) d\tau, \tag{3}$$

we can rewrite $J_k^{\sigma_k}$'s definition as

$$J_k^{\sigma_k} = \liminf_{H \to \infty} \frac{1}{H} \sum_{n_k=1}^{N_k(H)+1} \overline{C}_k(\tau_{n_k}) \tag{4}$$

Here, $\overline{C}_k(\tau_{n_k})$ is the total cost that arm k is expected to incur between $(n_k - 1)$-th and n_k-th plays according to schedule σ_k. Thus, $J_k^{\sigma_k}$ is just the average of these costs over the entire schedule. If the schedule stops playing arm k forever after some time t, $J_k^{\sigma_k}$ may be infinite.

Running a policy π amounts to sampling a joint schedule $\boldsymbol{\sigma} = \{\sigma_k\}_{k=1}^K$. Therefore, we define *policy cost* J^π as

$$\begin{aligned} J^\pi &\triangleq \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \pi}\left[\frac{1}{K} \sum_{k=1}^K J_k^{\sigma_k}\right] \\ &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \pi}\left[\frac{1}{K} \sum_{k=1}^K \left[\liminf_{H \to \infty} \frac{1}{H} \sum_{n_k=1}^{N_k(H)+1} \overline{C}_k(\tau_{n_k})\right]\right] \end{aligned} \tag{5}$$

Target policy class. Instead of considering all possible SYNCHRONIZATION policies as potential solutions, in this paper we focus on those whose sync-mode plays are *periodic*, with equal gaps between every two consecutive such plays of a

given arm. For arm k, we denote the length of these gaps as $1/r_k > 0$ length, r_k being a policy parameter for this arm and $\boldsymbol{r} \triangleq (r_k)_{k=1}^K$ being the joint parameter vector for all arms. Importantly, our policies do allow probe-mode arm plays but don't restrict how the time points for these plays are chosen. In particular, they may be chosen stochastically, as long the timings of sync-mode plays are deterministically periodic.

Formally, for a scheduled arm pull time point t_{n_k} in schedule σ_k, let $NextSync_{\sigma_k}(t_{n_k})$ be the next sync-mode play of arm k in σ_k, i.e., $t_{n'_k} = NextSync_{\sigma_k}(t_{n_k})$ if $t_{n_k} = \min\{t_{n''_k} \mid n''_k > n_k, (t_{n''_k}, l_{n''_k}) \in \sigma_k, l_{n''_k} = sync\}$. Then our target policy class is

$$\Pi = \{\pi(\boldsymbol{r}) \mid \forall[\boldsymbol{\sigma} \sim \pi(\boldsymbol{r}), k \in [K], \text{and } (t_{n_k}, l_{n_k}) \in \sigma_k \text{ s.t. } l_{n_k} = sync \text{ and } t_{n'_k} = NextSync_{\sigma_k}(t_{n_k})] \; t_{n'_k} - t_{n_k} = \frac{1}{r_k} \text{ for } r_k \in \boldsymbol{r}\} \quad (6)$$

Parameters $\boldsymbol{r}$ can be interpreted as rates at which arms are played in sync mode. For $\pi \in \Pi$, policy costs are uniquely determined by sync-mode play rates $\boldsymbol{r}$: although these parameters ignore the timing of probe plays, probe plays don't affect cost generation and therefore policy cost.

We let $J(\boldsymbol{r})$ denote $\pi(\boldsymbol{r}) \in \Pi$'s policy cost. Equation 5 implies that its cost functions $J(\boldsymbol{r})$ have a special form critical for our regret analysis in Section 5 – they are *convex*:

Lemma 1. *For any policy $\pi(\boldsymbol{r}) \in \Pi$,*

$$J(\boldsymbol{r}) = \frac{1}{K} \sum_{k=1}^{K} r_k \overline{C}_k \left(\frac{1}{r_k}\right). \quad (7)$$

$J(\boldsymbol{r})$ and $J_k(r_k) \triangleq r_k \overline{C}_k \left(\frac{1}{r_k}\right)$ for each $k \in [K]$ is convex and monotonically decreasing for $\boldsymbol{r} > \boldsymbol{0}$.

Proof. See the Appendix. ■

The convexity of the cost functions plays a crucial role in obtaining the regret bounds (Section 5) for the policy learning algorithm presented in Section 4.

Policy constraints. Naturally, we would like to find a $\pi(\boldsymbol{r}) \in \Pi$ that minimizes $J(\boldsymbol{r})$ (Equation 7). As described, however, Π has no such policy: note that $\lim_{\boldsymbol{r} \to \boldsymbol{\infty}} J(\boldsymbol{r}) = \boldsymbol{0}$, but no finite $\boldsymbol{r}$ attains this limit. However, in practical applications the rates r_k cannot be arbitrarily high individually or in aggregate, and are subject to several constraints. Therefore, in this paper we regard feasible r_k as bounded from above for all k by a universal bound r_{max}. Moreover, we assume that the sum of all arms' play rates may not exceed some value $B > 0$. E.g., in Web crawling B is commonly interpreted as a limit on crawl rate imposed by physical network infrastructure (Azar et al., 2018; Kolobov et al., 2019a; Upadhyay et al., 2020). Last but not least, valid r_k values may not be 0, since this implies never playing this arm after a certain time point. In applications such as Web crawling, this means abandoning a cached item (e.g., an indexed webpage) to grow arbitrarily stale, which is unacceptable, so we impose a minimum sync-mode play rate r_{min} on every arm. Note that since, by Assumption 1, every $c_k(\tau)$ is bounded for $\tau \geq 0$, every $J_k(r_k)$ (Lemma 1) is bounded as well.

Policy optimization. Thus, *if we knew cost processes $c_k(.)$*, we could use them to compute $\overline{C}_k(.)$ for all arms and would face the following optimization problem:

Problem 1 (SYNCHRONIZATION BANDIT instance)**.**

$$\textbf{\textit{Minimize:}}\ J(\boldsymbol{r}) = \frac{1}{K} \sum_{k=1}^{K} r_k \overline{C}_k \left(\frac{1}{r_k}\right) \quad (8)$$

$$\textbf{\textit{subject to:}}\ \boldsymbol{r} \in \mathcal{K} \triangleq \left\{\boldsymbol{r'} \in [r_{min}, r_{max}]^K \;\middle|\; ||\boldsymbol{r'}||_1 = B\right\}$$

Notice that this formulation implicitly assumes that the entire bandwidth B will be used for sync-mode arm plays – indeed, if the model is known, there is no need for probes.

As a side note, we remark that the class of periodic policies Π doesn't restrict Problem 1's solution quality compared to broader the class of deterministic open-loop policies. We state it here informally as a proposition, which we reformulate more precisely and prove in the Appendix A:

Proposition 1. *For a given K-armed* SYNCHRONIZATION BANDIT *instance (Problem 1), the optimal* periodic *policy $\pi^* \in \Pi$ has the same cost J^* as the optimal* general deterministic open-loop *policy under the same constraints.*

4. Online learning for cache synchronization

In reality we don't know the cost generation processes and can't solve the above optimization problem directly. Instead, we adopt an online perspective on SYNCHRONIZATION bandits. A key contribution of this paper that we present in this section is MIRRORSYNC (Algorithm 1), an algorithm that treats a SYNCHRONIZATION MAB as an online learning problem. A MIRRORSYNC agent can be viewed operates in rounds $\mathcal{T} = 1, 2, \ldots\ \mathcal{T}_{max}$, in each round "playing" a candidate policy parameter vector $\boldsymbol{r}^{\mathcal{T}}$, suffering a "loss" $\hat{J}^{\mathcal{T}} \sim J^{\mathcal{T}}(\boldsymbol{r}^{\mathcal{T}})$, and updating $\boldsymbol{r}^{\mathcal{T}}$ to a new vector $\boldsymbol{r}^{\mathcal{T}+1}$ as a result. As we show in Section 5, MIRRORSYNC has an adversarial regret of $O(\mathcal{T}_{max}^{2/3})$.

MIRRORSYNC's novelty is due to the fact that, despite superficial similarities to standard online learning, our setting is different from it in crucial ways, and MIRRORSYNC circumvents these differences:

(1) Although the agent can be viewed as suffering loss $\hat{J}_{\mathcal{T}}$, it doesn't *observe* this loss. By SYNCHRONIZATION MAB's assumptions, it observes only samples of instantaneous costs $c_k(.)$, and only when it plays arms. Existing techniques don't offer a way to estimate the gradient ∇J in this case. Moreover, even these impoverished observations take real-

world time to collect. **(2)** In online learning, regret analysis normally assumes ∇J to be bounded. This isn't quite the case in our model. While we could assume a bound on ∇J linear in $1/r_{min}$, it would be detrimental to the regret bound when $1/r_{min}$ is large. We show how MIRRORSYNC addresses challenge **(1)** in this section, and circumvent challenge **(2)** in Section 5.

A note on infinite vs. finite-horizon policies. The policy cost functions we derived in Section 3 describe the steady-state performance of a policy over an *infinite* time horizon. However, MIRRORSYNC's rounds are finite. Thus, the cost function J (Equation 7) that MIRRORSYNC uses as a basis for policy improvement is a proxy measure for the average costs that running MIRRORSYNC incurs in each round.

MIRRORSYNC operation. At a high level, MIRRORSYNC's main insight is allocating a small fraction of available bandwidth B, determined by input parameter ε (Algorithm 1), to probe-mode arm plays, while using the bulk $\left(\frac{1}{1+\varepsilon}\right) B$ of the bandwidth (line 2) to play in sync mode according to the current rates $\boldsymbol{r}$. MIRRORSYNC uses instantaneous cost samples obtained from both to estimate the gradient ∇J (lines 11 - 19) by individually estimating its partial derivatives (lines 23-25), which we denote as

$$\partial_k J \triangleq \frac{\partial J}{\partial r_k}$$

for short. At the end of each epoch, it does online mirror descent on these estimates to get a new sync-mode policy $\boldsymbol{r}$ (lines 20, 42-43).

In more detail, in the spirit of online learning, MIRRORSYNC assumes that at the start of each round all arms' cost generation processes have just been reset to $c_k(0)$, and restarts the time counter at $t = 0$ for every arm (line 9). (In practice, this assumption is unrealistic, and we lift it in another variant of MIRRORSYNC in Section 6.) Then, for every arm k, it schedules sync-mode plays at intervals $1/r_k$ (lines 29, 37-38) until the end of the current round, and attempts to insert one probe-mode play into each such interval (lines 33-36) independently with probability ε (line 32), at a point chosen uniformly at random over the interval's length (line 34). Executing the constructed schedule (line 13) yields cost samples that are used in the aforementioned gradient estimation, which, crucially, is unbiased:

Lemma 2. *For a rate vector $\boldsymbol{r} = (r_k)_{k=1}^K$ and a probability ε, suppose the agent plays each arm in sync mode $1/r_k$ time after that arm's previous sync-mode play, observing a sample of instantaneous cost $\hat{c}_k \sim c_k(1/r_k)$. Suppose also that in addition, with probability ε independently for each arm k, the agent plays arm k in probe mode at time $t \sim$ Uniform$[0, 1/r_k]$ after that arm's previous sync-mode play, observing a sample of instantaneous cost $\hat{c}_k^{(\varepsilon)} \sim c_k(t)$.*

Algorithm 1: MIRRORSYNC

Input: r_{min} – lowest allowable arm play rate
r_{max} – highest allowable arm play rate
B – bandwidth
ε – probability of probe-mode arm play
η – learning rate
$\mathcal{T}_{max}$ – number of rounds
Output: $\boldsymbol{r}$ – arm play rates.

1 $T_{round} \leftarrow 1/r_{min}$ // Round length
2 $\mathcal{K}_\varepsilon \leftarrow \left\{ \mathbf{x} \in [r_{min}, \frac{r_{max}}{1+\varepsilon}]^K \middle| ||\mathbf{r}||_1 = \frac{B}{1+\varepsilon} \right\}$
3 $\boldsymbol{r} \leftarrow arg\min_{\boldsymbol{x} \in \mathcal{K}_\varepsilon}$ **BarrierF**$(\boldsymbol{x})$ // Initialize play rates
4 **foreach** *round* $\mathcal{T} = 1, \ldots, \mathcal{T}_{max}$ **do**
5 // At the start of each round, all arms are assumed
6 // to be synchronized and time re-starts at 0.
7 **foreach** *arm* $k \in [K]$ **do**
8 // Construct a schedule $\sigma_k^{\mathcal{T}}$ for the $\mathcal{T}$-th round.
9 $t_{prev,k} \leftarrow 0$
10 $\sigma_k^{\mathcal{T}}, t_{prev,k} \leftarrow$ **ScheduleArmPlays**$(t_{prev,k}, r_k, T_{round})$
11 **foreach** *arm* $k \in [K]$ *simultaneously* **do**
12 // Sample costs by playing according to $\sigma_k^{\mathcal{T}}$
13 $(\hat{c}_{k,t_1}, \ldots, \hat{c}_{k,t_{|\sigma_k^{\mathcal{T}}|}}) \leftarrow Play(\sigma_k)$
14 **foreach** $n = 1, \ldots, |\sigma_k^{\mathcal{T}}|$ *and* $(t_n, l_n) \in \sigma_k^{\mathcal{T}}$ **do**
15 **if** l_n == $sync$ **then** $\hat{c}^{(\varepsilon)} \leftarrow none, \hat{c} \leftarrow \hat{c}_{k,t_n}$
16 **else** $\hat{c}^{(\varepsilon)} \leftarrow \hat{c}_{k,t_n}, \hat{c} \leftarrow \hat{c}_{k,t_{n+1}}, n \leftarrow n+1$
17 $\hat{g}_k \leftarrow$ $\frac{\textbf{GradJSample}(\hat{c}_k^{(\varepsilon)}, \hat{c}_k, r_k, \varepsilon)}{K}$
18 **break**
19 $\hat{\boldsymbol{g}}_{\boldsymbol{\mathcal{T}}} \leftarrow (\hat{g}_1, \ldots, \hat{g}_K)$
20 $\boldsymbol{r} \leftarrow$ **MirrorDescentStep**$(\eta, \mathcal{K}_\varepsilon, \hat{\boldsymbol{g}}_{\boldsymbol{\mathcal{T}}}, \boldsymbol{r})$
21 Return $\boldsymbol{r}$
22
23 **GradJSample**$(\hat{c}_k^{(\varepsilon)}, \hat{c}_k, r_k, \varepsilon)$:
24 **if** $\hat{c}_k^{(\varepsilon)}$ == $none$ **then** Return 0
25 **else** Return $\frac{1}{\varepsilon r_k}(\hat{c}_k^{(\varepsilon)} - \hat{c}_k)$
26
27 **ScheduleArmPlays**$(t_{prev,k}, r_k, interval\_len)$:
28 $\sigma_k \leftarrow (), \; n_k \leftarrow 0, t_0 \leftarrow t_{prev,k}$
29 **while** $t_{n_k} + 1/r_k < interval\_len$ **do**
30 $t_{prev\_sync} \leftarrow t_{n_k}$
31 $n_k \leftarrow n_k + 1$
32 $Probe \sim$ Bernoulli(ε)
33 **if** $Probe$ **then**
34 $t_{n_k} \sim$ Uniform$(t_{n_k-1}, t_{n_k-1} + 1/r_k)$
35 $\sigma_k \leftarrow Append(\sigma_k, (t_{n_k}, probe))$
36 $n_k \leftarrow n_k + 1$
37 $t_{n_k} \leftarrow t_{prev\_sync} + 1/r_k$
38 $\sigma_k \leftarrow Append(\sigma_k, (t_{n_k}, sync))$
39 $t_{prev,k} \leftarrow t_{n_k}$
40 Return $\sigma_k, t_{prev,k}$
41
42 **MirrorDescentStep**$(\eta, \mathcal{K}, \overline{\boldsymbol{g}}, \boldsymbol{r})$:
43 Return $\arg\min_{\boldsymbol{x} \in \mathcal{K}} \{\eta \langle \boldsymbol{x}, \overline{\boldsymbol{g}} \rangle + \textbf{DivF}(\boldsymbol{x}, \boldsymbol{r})\}$
44
45 **DivF**$(\boldsymbol{x}, \boldsymbol{r})$: Return $\sum_{k=1}^K -\log(x_k/r_k) + x_k/r_k - 1$
46
47 **BarrierF**$(\boldsymbol{r})$: Return $\sum_{k=1}^K -\log(r_k)$

Then for each k,

$$g_k \triangleq \begin{cases} 0 & \text{if } \neg \textit{Bernoulli}(\varepsilon) \\ \frac{1}{\varepsilon r_k K}(\hat{c}_k^{(\varepsilon)} - \hat{c}_k) & \text{if } \textit{Bernoulli}(\varepsilon) \end{cases}$$

is an unbiased estimator of $\partial_k J(r_k)$.

Proof. See the Appendix. ■

In one round, MIRRORSYNC may get several gradient estimates for a given arm, in which case it takes the first one (line 18). To get a regret bound, however, it is crucial to ensure that for each arm MIRRORSYNC receives *at least one* such estimate per round, even if the estimate is 0 (line 18). This is why we set the length of each round to be $1/r_{min}$ (line 1) — the largest value $1/r_k$ and hence the longest time that MIRRORSYNC may have to wait in order to get a cost sample at $1/r_k$.

5. Regret analysis

We generalize our stochastic setting to an adversarial problem and prove an adversarial regret bound of order $\mathcal{O}\left(T^{\frac{2}{3}}\right)$. This means that the cost distributions c_k and all derived quantities ($\bar{c}_k$, $\bar{C}_k$, J_k) need not be non-stationary from one round to the next. The cost distributions and derived functions at round $\mathcal{T}$ are denoted by $c_k^{\mathcal{T}}$, $\bar{c}_k^{\mathcal{T}}$, $\bar{C}_k^{\mathcal{T}}$, $J_k^{\mathcal{T}}$ and can be chosen by an oblivious adversary ahead of time.

Regret. We define the regret with respect to a fixed schedule $\boldsymbol{r} \in [0, \infty)^d$ by

$$\mathrm{Reg}(\boldsymbol{r}) \triangleq \mathbb{E}\left[\sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} J^{\mathcal{T}}(\boldsymbol{r}^{\mathcal{T}})\right] - \sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} J^{\mathcal{T}}(\boldsymbol{r}),$$

where the expectation is over the randomness of observed costs $\hat{c}_k$ and $\boldsymbol{r}^{\mathcal{T}}$ is the choice of the algorithm in round $\mathcal{T}$. Our goal is to compete with the best possible schedule $\boldsymbol{r}^*$ using the full available budget:

$$\mathrm{Reg} = \mathrm{Reg}(\boldsymbol{r}^*), \text{ where } \boldsymbol{r}^* \triangleq \min_{\boldsymbol{r} \in \mathcal{K}_0} \sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} J^{\mathcal{T}}(\boldsymbol{r}),$$

where $\mathcal{K}_0$ is $\mathcal{K}_\varepsilon$ (line 2 of Algorithm 1) with $\varepsilon = 0$.

Since we are not be able to obtain any information on the function value or gradient of $J^{\mathcal{T}}$ without an allocated exploration, we also define the best possible schedule $\boldsymbol{r}_\varepsilon^*$ given an exploration constrained by ε (lines 32 - 34 of Algorithm 1):

$$\boldsymbol{r}_\varepsilon^* \triangleq \arg\min_{\boldsymbol{r} \in \mathcal{K}_\varepsilon} \sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} J^{\mathcal{T}}(\boldsymbol{r}).$$

The regret can be decomposed into

$$\mathrm{Reg} = \underbrace{\mathrm{Reg}(\boldsymbol{r}_\varepsilon^*)}_{\text{in-policy regret}} + \underbrace{\sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} \left(J^{\mathcal{T}}(\boldsymbol{r}_\varepsilon^*) - J^{\mathcal{T}}(\boldsymbol{r}^*)\right)}_{\text{exploration gap}},$$

which we bound separately.

In-policy regret. The problem is an instance of online learning, but online learning literature typically assumes that the gradients of the objective functions $\nabla J^{\mathcal{T}}$ are uniformly bounded w.r.t. some norm. Our setting differs in a key aspect: the gradients $\partial_k J(\boldsymbol{r})$ scale proportionally to r_k^{-1}.

A naive solution would be to bound $\partial_k J(\boldsymbol{r}) \lesssim r_{min}^{-1}$ and use gradient descent. However, the regret would scale with r_{min}^{-1}, which might be prohibitively large.

We show that mirror descent with a carefully chosen potential, namely the *log* barrier $F(\boldsymbol{r}) = \sum_{k=1}^K \log(r_k)$, adapts to the geometry of the gradients and replaces the polynomial dependency on r_{min}^{-1} by a log dependency.

Theorem 5.1. *For any sequence of convex functions $(J^{\mathcal{T}})_{\mathcal{T}=1}^{\mathcal{T}_{max}}$ and learning rate $0 < \eta < \frac{K\varepsilon}{2U}$, the in-policy regret of* MIRRORSYNC *is bounded by*

$$\mathrm{Reg}(\boldsymbol{r}_\varepsilon^*) \leq \frac{K}{\eta} \log\left(\frac{B}{r_{min} K}\right) + \eta \frac{U^2}{\varepsilon K} \mathcal{T}_{max}.$$

Proof. See the Appendix. ■

Exploration Gap. We show that the exploration gap scales proportionally to ε and is independent of r_{min}^{-1}. On first sight, this bound is surprising because the exploration gap should be approximately $\langle \sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} \nabla J^{\mathcal{T}}(\boldsymbol{r}^*), \boldsymbol{r}^* - \boldsymbol{r}_\varepsilon^* \rangle$ and the gradients $\nabla J_k^{\mathcal{T}}(\boldsymbol{r}^*)$ could be unbounded (i.e. of order $\boldsymbol{r}_{min}^{-1}$). The high-level idea behind the following lemma is the observation that at the optimal point $\boldsymbol{r}^*$, the gradients in all coordinates must coincide and hence the gradient cannot be of order r_{min}^{-1} even if $r_k^* = r_{min}$.

Lemma 3. *The exploration gap is bounded by*

$$\sum_{\mathcal{T}=1}^{\mathcal{T}_{max}} \left(J^{\mathcal{T}}(\boldsymbol{r}_\varepsilon^*) - J^{\mathcal{T}}(\boldsymbol{r}^*)\right) \leq 2\varepsilon U \mathcal{T}_{max}.$$

Proof. See the Appendix. ■

Finally we are ready to present the main result.

Corollary 1. *The regret of* MIRRORSYNC *with $\eta = \frac{K}{U}\sqrt{\log\left(\frac{B}{r_{min}K}\right)\frac{\varepsilon}{\mathcal{T}_{max}}}$ and $\varepsilon = \mathcal{T}_{max}^{-\frac{1}{3}} \log^{\frac{1}{3}}\left(\frac{B}{r_{min}K}\right)$ is bounded for any $\mathcal{T}_{max} > 8 \log\left(\frac{B}{r_{min}K}\right)$ by*

$$\mathrm{Reg} \leq 3U \mathcal{T}_{max}^{\frac{2}{3}} \log^{\frac{1}{3}}\left(\frac{B}{r_{min}K}\right). \tag{9}$$

Proof. The choice of η, ε and the bound on $\mathcal{T}_{max}$ ensure that we can apply Theorem 5.1 to bound the in-policy regret. The in-policy regret simplifies to

$$\mathrm{Reg}(\boldsymbol{r}_\varepsilon^*) \leq 2U\sqrt{\frac{\mathcal{T}_{max}}{\varepsilon} \log\left(\frac{B}{r_{min}K}\right)}.$$

Adding the exploration gap from Lemma 3 and substituting the value for ε completes the proof. ■

6. Making MIRRORSYNC practical

Although MIRRORSYNC lends itself to theoretical analysis, several design choices make its vanilla version impractical. **(1)** MIRRORSYNC assumes that all arms are synchronized "for free" at the start of each round so that each round starts in the same "state", which is unrealistic. **(2)** MIRRORSYNC waits until the end of each $1/r_{min}$-long round to update all arms' play rates simultaneously, which could be months in applications like Web crawling. **(3)** MIRRORSYNC's further source of inefficiency is that even if arm k has produced several $\partial_k J(r_k)$ samples in a given round, MIRRORSYNC uses only one of them. ASYNCMIRRORSYNC (Algorithm 2), which can be viewed as a practical implementation of MIRRORSYNC, mitigates these weaknesses of MIRRORSYNC.

In contrast to MIRRORSYNC, which performs updates in rigidly defined rounds, ASYNCMIRRORSYNC updates policy according to a user-specified schedule $\mathcal{S}$ (see Algorithm 2's inputs). The length of inter-update periods is unimportant for ASYNCMIRRORSYNC, unlike for MIRRORSYNC (line 1, Alg. 1), due to a major difference between the two algorithms. MIRRORSYNC aims to update all arms' parameters *synchronously* at the end of each round and makes the rounds very long to *guarantee* that each arm has generated at least one gradient estimate by the end of each round. In the meantime, ASYNCMIRRORSYNC does updates *asynchronously*, performing mirror descent at an update time $t_i^{(upd)} \in \mathcal{S}$ only on those arms that happen to have generated at least one new gradient sample since the previous update time $t_{i-1}^{(upd)}$ (lines 26-33). ASYNCMIRRORSYNC does these local updates while respecting the global constraint B by using the sum of current play rates or arms that are about to be updated as a local constraint (line 32). Thus, ASYNCMIRRORSYNC doesn't need to make inter-update intervals excessively long and doesn't suffer from issue (1).

As a side note, the reason MIRRORSYNC's regret bound in Corollary 1 has no linear dependence on $1/r_{min}$ is because it characterizes regret in terms of the number of *rounds*, not wall-clock time. Nonetheless, this dependency matters empirically, and obtaining a wall-clock-time regret bound that is free from it is an interesting theoretical problem.

ASYNCMIRRORSYNC's asynchronous update mechanism also removes the need for "free" simultaneous sync-mode play of all arms after each round (2). Recall that before each sync-mode play of arm k with probability ε we can play arm k another time, and so far we have chosen to do so in probe mode. The **ScheduleArmPlays** routine (line 27, Alg. 1) that both MIRRORSYNC and ASYNCMIRRORSYNC rely on attempts this (lines 32-33, Alg. 1) for every sync-mode arm play *except* the first arm play of each round. ASYNCMIRRORSYNC takes advantage these unused chances

Algorithm 2: ASYNCMIRRORSYNC

Input: r_{min} – lowest allowable arm play rate
r_{max} – highest allowable arm play rate
B – bandwidth
ε – probability of probe-mode arm play
η – learning rate
T_{max} – (wall-clock) time horizon
$\mathcal{S} = (t_1^{(upd)}, t_2^{(upd)}, ...)$ – update schedule
Output: $\boldsymbol{r}$ – arm play rates.

1 $\mathcal{K}_\varepsilon \leftarrow \left\{ \mathbf{x} \in [r_{min}, \frac{r_{max}}{1+\varepsilon}]^K \middle| ||\mathbf{r}||_1 = \frac{B}{1+\varepsilon} \right\}$
2 $\boldsymbol{r} \leftarrow arg\min_{\boldsymbol{x} \in \mathcal{K}_\varepsilon}$ **BarrierF**$(\boldsymbol{x})$ // Initialize play rates
3 $t_{now} \leftarrow 0$ // current time
4 // Extend arms' schedules σ_k until the next update time
foreach *arm* $k \in [K]$ **do**
5 $t_{prev,k} \leftarrow t_{now}$
6 $\sigma_k, t_{prev,k} \leftarrow$ **ScheduleArmPlays**$(t_{prev,k}, r_k, t_1^{(upd)})$
7 // t_{now} is incremented continuously
8 **while** $t_{now} \leq T_{max}$ **do**
9 // Play each arm k's current σ_k, record cost samples
10 **foreach** *arm* $k \in [K]$ *simultaneously* **do**
11 $(\hat{c}_{k,t_1}, \ldots, \hat{c}_{k,t_{|\sigma_k|}}) \leftarrow Play(\sigma_k)$
12 // If now is the next update time i ...
13 **if** t_{now}==$t_i^{(upd)}$ *for some* $t_i^{(upd)} \in \mathcal{S}$ **then**
14 $\mathcal{A}_i \leftarrow \emptyset$ // Set of arms that will be updated now
15 // Collect cost samples since prev. update time
16 **foreach** *arm* $k \in [K]$ **do**
17 **foreach** $n, (t_n, l_n) \in \sigma_k \mid t_n \geq t_{i-1}^{(upd)}$ **do**
18 **if** l_n == *sync* **then**
19 $\hat{c}^{(\varepsilon)} \leftarrow none, \hat{c} \leftarrow \hat{c}_{k,t_n}$
20 **else** // This was a probe play
21 $\hat{c}^{(\varepsilon)} \leftarrow \hat{c}_{k,t_n}, \hat{c} \leftarrow \hat{c}_{k,t_{n+1}}$
22 $n \leftarrow n+1$
23 // Estimate $\partial_k J$ per Lemma 2 (Alg. 1,
24 // lines 23-25) using collected samples
25 $\hat{g}_{n,k} \leftarrow$ **GradJSample**$(\hat{c}_k^{(\varepsilon)}, \hat{c}_k, r_k, \varepsilon)$
26 **if** *we got at least one* $\hat{g}_{n,k}$ *for arm* k **then**
27 $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{k\}$
28 $\overline{g}_k \leftarrow Avg(\{\hat{g}_{n,k} \mid t_n \geq t_{i-1}^{(upd)}\})$
29 // Normalize new grad. estimates
30 $\overline{\boldsymbol{g}}_{\boldsymbol{i}} \leftarrow \frac{1}{|\mathcal{A}_i|}(\overline{g}_k)_{k \in \mathcal{A}_i}$
31 // Now, update play rates *only* for arms in $\mathcal{A}_i$
32 $\mathcal{K}_{\varepsilon,i} \leftarrow \left\{ \mathbf{x} \in [r_{min}, \frac{r_{max}}{1+\varepsilon}]^{|\mathcal{A}_i|} \middle| ||\mathbf{r}||_1 = \frac{\sum_{k \in \mathcal{A}_i} r_k}{1+\varepsilon} \right\}$
33 $\boldsymbol{r}_{\boldsymbol{\mathcal{A}_i}} \leftarrow$ **MirrorDescentStep**$(\eta, \mathcal{K}_{\varepsilon,i}, \overline{\boldsymbol{g}}_{\boldsymbol{i}}, \boldsymbol{r}_{\boldsymbol{\mathcal{A}_i}})$
34 **foreach** *arm* $k \in [K]$ **do**
35 // Extend sched. σ_k until next update time.
36 **if** *Bernoulli*(ε) **then** $t_{prev,k} \leftarrow t_{now}$
37 **else** $t_{prev,k} \leftarrow \max\{t_{prev,k} + \frac{1}{r_k}, t_{now}\}$
38 $\sigma_k \leftarrow Append(\sigma_k, (t_{prev,k}, sync))$
39 $\sigma'_k, t_{prev,k} \leftarrow$ **ScheduleArmPlays**$(t_{prev,k}, r_k, t_{i+1}^{(upd)})$
$\sigma_k \leftarrow Append(\sigma_k, \sigma'_k)$
40 Return $\boldsymbol{r}$

to schedule *sync-mode* plays, which reset cost generation for some fraction of arms (line 36, Alg. 2). For the remaining arms, it simply waits until their next sync-mode play (line 37, Alg. 2) to start estimating the new gradient.

Last but not least, AsyncMirrorSync improves the efficiency of updates themselves. It employs all gradient samples we get for an arm between updates, and averages them to reduce estimation variance (lines 25, 28), thereby rectifying MirrorSync's weakness (3).

7. Related work

There are several existing models superficially related to but fundamentally different from synchronization MABs.

In *maintenance job scheduling* (Bar-Noy et al., 1998), as in our setting, each machine (arm) has an associated operating cost per time unit that increases since the previous maintenance, and performing a maintenance reduces this cost temporarily. However, the agent knows all arms' cost functions and always observes the machines' running costs.

Upadhyay et al. (2018) describe a model for maximizing a long-term reward that is a function of two general marked temporal point processes. This model is more general than synchronization MAB in some ways (e.g., not assuming cost process monotonicity) but allows controlling the policy process's rate only via a policy cost regularization term and provides no performance guarantees.

Recharging bandits (Immorlica & Kleinberg, 2018), like synchronization MABs, have arms with non-stationary payoffs: the expected arm reward is a convex increasing function of time since the arm's last play. In spite of this similarity, recharging bandits and other MABs with time-dependent payoffs (Heidari et al., 2016; Levine & Crammer, 2017; Cella & Cesa-Bianchi, 2020) make the common assumptions that a reward is generated only when an arm is played and that the agent observes all generated rewards. As a result, their analysis is different from ours. In general, payoff non-stationarity has been widely studied in two broad bandit classes. Restless bandits (Whittle, 1988) allow an arm's reward distributions to change, but only *independently of* when the arm is played. Rested bandits (Gittins, 1979) also allow an arms' reward distribution changes, but only *when* the arm is played. Synchronization MABs belong to neither class, since their arms' instantaneous costs change both independently and a result of arms being played.

8. Empirical evaluation

The goal of our experiments was to evaluate (1) the relative performance of MirrorSync and AsyncMirrorSync, given that MirrorSync assumes "free" arm resets at the beginning of each round and AsyncMirrorSync doesn't, and (2) the relative performance of AsyncMirrorSync and its version with projected stochastic gradient descent (PSGD) instead of mirror descent, which we denote as AsyncPSGDSync. The choice of mirror descent instead of PSGD was motivated by the intuition that with mirror descent MirrorSync would achieve lower regret than with PSGD (see Section 5). In the experiments, we verify this intuition empirically. Before analyzing the results, we describe the details of our experiment setup.

Problem instance generation. Our experiments in Figures 1 and 2 were performed on synchronization MAB instances generated as follows. Recall from Sections 2 and 3 that a synchronization MAB instance is defined by:

- r_{min}, the lowest allowed arm play rate
- r_{max}, the highest allowed arm play rate
- B, the highest allowed *total* arm play rate
- K, the number of arms
- $\{c_k(\tau)\}_{k=1}^K$, a set of cost-generating processes — time-dependent distributions of instantaneous costs, one process for each arm k.

For all problem instances in the experiments, r_{min}, r_{max}, B, and K were as in Table 1 in Appendix B. The set $\{c_k(\tau)\}_{k=1}^K$ of cost-generating processes was constructed randomly for each instance. In all problem instances, each arm had a distribution over time-dependent cost functions in the form of polynomials $c_k(\tau) = a_k\tau^{p_k}$, where $p_k \in (0, 1)$ was chosen at instance creation time and a_k sampled at run time as described in Appendix B. Note that MirrorSync's regret (Equation 9) depends on the cost cap U. While our polynomial cost functions are unbounded in general, they are bounded within the $[r_{min}, r_{max}]$ constraint region we are using (Table 1 in Appendix B). Within this constraint region, these cost functions are equivalent to $\min\{a_k\tau^{p_k}, U\}$, where $U = 40$.

In Appendix C we also describe a different, Poisson process-based family of cost-generating processes, and present experimental results obtained on it in Figures 3 and 4 in that section. Despite that process family being very distinct from the polynomial one, the results are qualitatively similar to those in Figures 1 and 2.

Implementation details. We implemented MirrorSync, AsyncMirrorSync, and AsyncPSGDSync, along with a problem instance generator that constructs synchronization MAB instances as above, in Python. The implementation, available at https://github.com/microsoft/Optimal-Freshness-Crawl-Scheduling, relies on `scipy.optimize.minimize` for convex constrained optimization in order to update the play rates $\boldsymbol{r}$ (lines 20, 42 of Alg. 1, line 33 of Alg. 2). Other convex optimizers are possible as well. The experiments were performed on a Windows 10 laptop with 32GB RAM with 8 Intel 2.3GHz i9-9980H CPU cores.

Hyperparameter tuning. Running the algorithms required

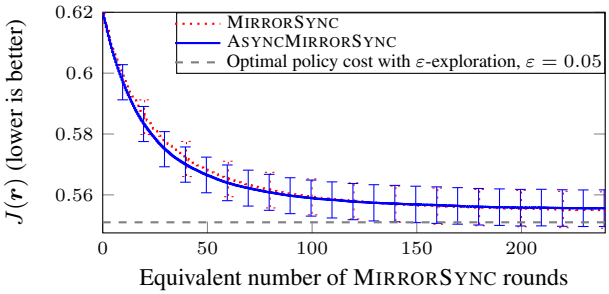

Figure 1. MIRRORSYNC's and ASYNCMIRRORSYNC's convergence. The two algorithms exhibit nearly identical convergence behavior using tuned hyperparameters. However, ASYNCMIRRORSYNC works without assuming the "free" arm resets after arms' parameter updates that MIRRORSYNC relies on.

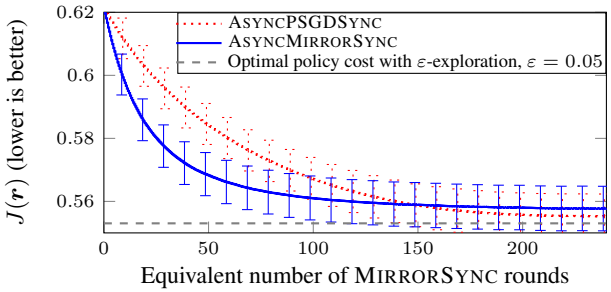

Figure 2. Asynchronous version of MIRRORSYNC with mirror descent vs. with projected SGD. The use of mirror descent with the *log* barrier function in MIRRORSYNC was key to constructing the regret bounds in Section 5. Empirically, although ASYNCMIRRORSYNC and ASYNCPSGDSYNC eventually converge to similar-quality policies, ASYNCMIRRORSYNC discovers good policies faster, as MIRRORSYNC's theory predicts.

choosing values for the following parameters:

- **Learning rate η.** As in other learning algorithms, choosing a good value for η for each of the three algorithms was critical for their convergence behavior.
- **Length l_{upd_round} of intervals between ASYNCMIRRORSYNC's and ASYNCPSGDSYNC's play rate updates.** Recall that unlike MIRRORSYNC, which updates all play rates simultaneously after every $1/r_{min}$ time units, ASYNCMIRRORSYNC and ASYNCPSGDSYNC update the model parameters according to a user-provided schedule. While the schedule doesn't necessarily have to be periodic, in the experiments it was, with l_{upd_round} being the inter-update interval length. Intuitively, l_{upd_round} influences the average number of arms updated during each update attempt and the variance of gradient estimates: the larger l_{upd_round}, the more gradient samples ASYNCMIRRORSYNC and ASYNCPSGDSYNC can be expected to average for the upcoming update (line 28 of Alg. 2). In this respect, l_{upd_round}'s role resembles that of minibatch size in minibatch SGD.
- **Exploration parameter ε.** Theory provides a horizon-dependent guidance for setting ε for MIRRORSYNC (Corollary 1) but not for ASYNCMIRRORSYNC and ASYNCPSGDSYNC. For comparing *relative* convergence properties of MIRRORSYNC, ASYNCMIRRORSYNC, and ASYNCPSGDSYNC, we fixed $\varepsilon = 0.05$ for all of them.

ASYNCMIRRORSYNC's and ASYNCPSGDSYNC's performance is determined by a combination of η and l_{upd_round}, so we optimized them together using grid search. Please see Appendix B for more details.

Experiment results. Figures 1 and 2 compare the performance of MIRRORSYNC vs. ASYNCMIRRORSYNC and ASYNCMIRRORSYNC vs. ASYNCPSGDSYNC, respectively. The figure captions highlight important patterns we observed. The plots were obtained by running the respective pairs of algorithms on 150 problem instances generated as above, measuring the policy cost J after every update, and averaging the resulting curves across these 150 trials.

In each trial, all algorithms were run for the amount of simulated time *equivalent to* 240 MIRRORSYNC rounds (see Figure 1's and 2's x-axis). However, note that the number of updates performed by ASYNCMIRRORSYNC and ASYNCPSGDSYNC was larger than 240. Specifically, a MIRRORSYNC's update round is always of length $1/r_{min}$ time units, but for ASYNCMIRRORSYNC and ASYNCPSGDSYNC it is l_{upd_round} units, so for every MIRRORSYNC update round, they perform $(1/r_{min})/l_{upd\_round}$ updates. Although more frequent model updates is itself a strength of the asynchronous algorithms, their main practical advantage is independence of MIRRORSYNC's "free arm resets" assumption.

9. Conclusion

This paper presented SYNCHRONIZATION MABs, a bandit class where all arms generate costs continually, independently of being played, and the agent observes an arm's stochastic instantaneous cost only when it plays the arm. We proposed an online learning approach for this setting, called MIRRORSYNC, whose novelty is in estimating the policy cost gradient without directly observing the policy cost function and without having a closed-form expression for it. Moreover, we derived an $O(T^{\frac{2}{3}})$ adversarial regret bound for MIRRORSYNC without explicitly requiring the gradients to be bounded. We also presented ASYNCMIRRORSYNC, a practical version of MIRRORSYNC that lifts the latter's idealizing assumptions. The key insight behind all these contributions is that the use of mirror descent for policy updates in SYNCHRONIZATION MABs enables much faster convergence than gradient descent would. Our experiments confirmed this insight empirically.

Acknowledgements. We would like to thank Nicole Immorlica (Microsoft Research) and the anonymous reviewers for their helpful comments and suggestions regarding this work.

References

Azar, Y., Horvitz, E., Lubetzky, E., Peres, Y., and Shahaf, D. Tractable near-optimal policies for crawling. *Proceedings of the National Academy of Sciences (PNAS)*, 2018.

Bar-Noy, A., Bhatia, R., Naor, J., and Schieber, B. Minimizing service and operation costs of periodic scheduling. In *SODA*, pp. 11–20, 1998.

Bright, L., Gal, A., and Raschid, L. Adaptive pull-based policies for wide area data delivery. *ACM Transactions on Database Systems (TODS)*, 31(2):631–671, 2006.

Bubeck, S. *Convex Optimization: Algorithms and Complexity*. Foundations and Trends in Machine Learning, 2016.

Cella, L. and Cesa-Bianchi, N. Stochastic bandits with delay-dependent payoffs. In *AISTATS*, 2020.

Cho, J. and Garcia-Molina, H. Synchronizing a database to improve freshness. In *ACM SIGMOD International Conference on Management of Data*, 2000.

Cho, J. and Garcia-Molina, H. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426, 2003a.

Cho, J. and Garcia-Molina, H. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3): 256–290, 2003b.

Cho, J. and Ntoulas, A. Effective change detection using sampling. In *VLDB*, 2002.

Gal, A. and Eckstein, J. Managing periodically updated data in relational databases. *Journal of ACM*, 48:1141–1183, 2001.

Gittins, J. C. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2), 1979.

Heidari, H., Kearns, M., and Roth, A. Tight policy regret bounds for improving and decaying bandits. In *AISTATS*, 2016.

Immorlica, N. and Kleinberg, R. Recharging bandits. In *FOCS*, 2018.

Kolobov, A., Peres, Y., Lu, C., and Horvitz, E. Staying up to date with online content changes using reinforcement learning for scheduling. In *NeurIPS*, 2019a.

Kolobov, A., Peres, Y., Lubetzky, E., and Horvitz, E. Optimal freshness crawl under politeness constraints. In *SIGIR*, 2019b.

Levine, N. and Crammer, K. Rotting bandits. In *NIPS*, 2017.

Nemirovsky, A. and Yudin, D. *Problem complexity and method efficiency in optimization*. Wiley, 1983.

Robbins, H. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

Upadhyay, U., De, A., and Gomez-Rodriguez, M. Deep reinforcement learning of marked temporal point processes. In *NeurIPS*, 2018.

Upadhyay, U., Busa-Fekete, R., Kotlowski, W., Pal, D., and Szorenyi, B. Learning to crawl. In *AAAI*, 2020.

Whittle, P. Restless bandits: Activity allocation in a changing world. *Applied Probability*, 25(A):287–298, 1988.

Wolf, J. L., Squillante, M. S., Yu, P. S., Sethuraman, J., and Ozsen, L. Optimal crawling strategies for web search engines. In *WWW*, 2002.