Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

The Queue Method: Handling Delay, Heuristics, Prior Data, and Evaluation in Bandits

Travis Mandel[1], **Yun-En Liu**[1], **Emma Brunskill**[2], **and Zoran Popović**[1]
[1]Center for Game Science, Computer Science & Engineering, University of Washington
[2]School of Computer Science, Carnegie Mellon University
{tmandel, yunliu, zoran}@cs.washington.edu, ebrun@cs.cmu.edu

Abstract

Current algorithms for the standard multi-armed bandit problem have good empirical performance and optimal regret bounds. However, real-world problems often differ from the standard formulation in several ways. First, feedback may be delayed instead of arriving immediately. Second, the real world often contains structure which suggests heuristics, which we wish to incorporate while retaining strong theoretical guarantees. Third, we may wish to make use of an arbitrary prior dataset without negatively impacting performance. Fourth, we may wish to efficiently evaluate algorithms using a previously collected dataset. Surprisingly, these seemingly-disparate problems can be addressed using algorithms inspired by a recently-developed queueing technique. We present the Stochastic Delayed Bandits (SDB) algorithm as a solution to these four problems, which takes black-box bandit algorithms (including heuristic approaches) as input while achieving good theoretical guarantees. We present empirical results from both synthetic simulations and real-world data drawn from an educational game. Our results show that SDB outperforms state-of-the-art approaches to handling delay, heuristics, prior data, and evaluation.

Introduction

A key part of AI systems is the ability to make decisions under uncertainty. Multi-armed bandits (Thompson 1933) are a common framework for deciding between a discrete number of choices with uncertain payoffs. For example, we might wish to select the best ad to show on a webpage to maximize revenue, or we may have several different educational strategies and we want to select the one that maximizes learning. However, we do not want to decouple exploration and exploitation: we want to use the feedback we get from pulling one arm to inform the next pull.

The standard bandit problem has been well studied, resulting in algorithms with near-optimal performance both in theory (Auer, Cesa-Bianchi, and Fischer 2002) and empirically (Vermorel and Mohri 2005). However, most real-world problems do not match the bandit framework exactly. One difference is delay: if we present a problem to a student, other students who we need to interact with may arrive before the original student has finished. Also, when we have distributed systems with thousands of incoming users, real-time policy updates may be infeasible; instead, one typically launches a fixed (possibly stochastic) policy for a batch of users, updating it only after a period of time. These related problems, delayed feedback and batch updating, pose challenges for typical bandit algorithms theoretically (Desautels, Krause, and Burdick 2012) and empirically (Chapelle and Li 2011). Although stochastic algorithms perform best empirically in the face of delay (Chapelle and Li 2011), deterministic algorithms have the best theoretical guarantees (Joulani, Gyorgy, and Szepesvari 2013).

Second, we may know heuristic algorithms that do not possess good theoretical guarantees, but tend to perform well. For example, encouraging an algorithm to be more exploitative often results in good short-term performance (Chapelle and Li 2011), but may cause it to eventually settle on a sub-optimal arm. Or, we can make a structural assumption that similar arms share information which enables good performance if this assumption is true (Scott 2010), but if false results in poor performance. Ideally, we could incorporate a heuristic to help improve performance while retaining strong theoretical guarantees.

Third, we may have a prior data set consisting of information from a subset of arms, such as a website which decides to add new ads to help improve revenue. We want to make use of the old ad data without being overly biased by it, since the data can be drawn from an arbitrary arm distribution and thus possibly hurt performance. Ideally, we want a method that guarantees good black-box algorithms will perform well (or even better) when leveraging a prior dataset.

Finally, if we want to compare different bandit algorithms (and associated parameter settings) on a real application, running them online is often prohibitively expensive, especially when running an algorithm requires interacting with people. One could build a simulator, possibly using collected data, but simulators will often lack strong guarantees on their estimation quality. A preferred approach is data-driven offline evaluation, especially if it has strong theoretical guarantees such as unbiasedness. However, previously-proposed unbiased estimators need to know the original sampling policy (Li et al. 2011; Dudík et al. 2012) and tend to be sample-inefficient. We would prefer a more efficient, theoretically sound estimator that does not need to know the original sampling distribution.

In this paper, we present solutions to these four problems, all of which are based on the queue method. The queue method was inspired by the recent work of Joulani et al. 2013, and is conceptually simple: instead of allowing our black-box bandit algorithm to actually pull from arm i in the true environment, we draw a reward from a queue associated with i. This allows us to place arbitrary samples in the queues while ensuring that the black-box algorithm behaves as it would if it was actually pulling from the true environment. In this work, we show how this method can be used to produce good solutions to four problems. First, we present Stochastic Delayed Bandits (SDB), an algorithm that improves over QPM-D by taking better advantage of stochastic algorithms in the face of delay. Next, we show how one can use SDB to take advantage of heuristics while retaining strong theoretical guarantees. We also show how SDB can be robust to prior data. Last, we present a conditionally unbiased queue-based estimator. Our estimator does not require any knowledge of data collection policy, and allows us to evaluate algorithms for many more pulls compared to previously-proposed unbiased estimators. In addition to theoretical analysis, we demonstrate good empirical performance on both synthetic simulations and real world data drawn from thousands of students playing an educational game.

Background

We consider stochastic multi-armed bandit problems, in which there are a set of N arms labeled as integers $i \in \{1, \dots, N\}$, each of which has an unknown reward distribution with mean μ_i. In the standard online (i.e. non-delayed) problem formulation, at each timestep $t \in \{1, \dots, T\}$ the agent pulls arm $i \in \{1, \dots, N\}$ and receives a reward r_t drawn iid from arm i's reward distribution. The goal of the agent is to maximize its expected reward, or equivalently minimize its expected cumulative regret $\mathbb{E}[R(T)] = \sum_{i=1}^{N} \Delta_i L_{i,T}$, where $\Delta_i = (\max_j \mu_j) - \mu_i$ and $L_{i,T}$ is the number of times the algorithm has played arm i at time T.

We additionally consider versions of the problem with some *delay*: the reward for a pull at time t becomes available to the algorithm only at some time $t' > t$. We assume there is some unknown, arbitrary process which generates delay times, but similar to Joulani et al. 2013 we assume the delay is bounded by some unknown maximum τ_{max}.

We want to draw a distinction between *online updating* and *batch updating* in the presence of delay. In the *online updating* case, the algorithm can explicitly control which arm it pulls at every time t, even if it receives no feedback at time t. In the *batch updating* case, the algorithm must specify a single distribution over arm pulls to be run for an entire batch of pulls, and can only update it once the batch is complete. Batch updating makes more sense for distributed systems where deploying new policies is challenging to perform online (but can be done every hour or night). To simplify the analysis we assume that we observe the rewards of all arms pulled during the batch once the batch completes; thus the maximum delay, τ_{max}, is equal to the maximum batch size[1]. We again assume this quantity is unknown.

Stochastic Delayed Bandits

One central question underlies our four problems: how can we give arbitrary data to an arbitrary bandit algorithm BASE such that BASE behaves exactly as it would if it were run online? Such a procedure would allow us to feed delayed samples, prior datasets, and samples collected by a heuristic to BASE without impacting its behavior, thus retaining theoretical guarantees. To solve this problem efficiently, we must heavily rely on the stochastic assumption: each reward from arm i is drawn iid. As observed by Joulani et al. 2013, this assumption allows us to put collected rewards into queues for each arm. When BASE selects an arm I we simply draw an item from the queue I, allowing BASE to behave as it would if it pulled arm I in the true environment. This method defines approaches we call queue-based bandit algorithms (see Algorithm 1).

However, the more challenging question is how to sample to put items in the queues. If we want to guarantee good regret with respect to the BASE algorithm, this sampling cannot be truly arbitrary; instead, the behavior of BASE must influence the sampling distribution. In the online case, the obvious solution is to add a sample to a queue as soon as BASE wishes to pull an arm with an empty queue, but it is more challenging to determine what to do when rewards are returned with some delay. Joulani et al. 2013 addressed this problem with their QPM-D meta-algorithm, which continues to put items in the requested empty queue until they receive at least one reward for that arm (namely Algorithm 1 with GETSAMPLINGDIST being $q_I = 1; q_{i \neq I} = 0$). They proved that QPM-D achieves the best-known bound on stochastic bandit regret under delay.

Although QPM-D has a good regret bound, as we will see in our experiments, its empirical performance is somewhat limited. This is mostly due to the fact that it is a deterministic algorithm, while algorithms producing stochastic arm policies have been shown to perform much better in practice when delay is present (Chapelle and Li 2011), since they make better use of the time between observations. For example, consider a batch update setting with a very long initial batch: a stochastic policy can get information about all arms, while a deterministic policy uses the entire batch to learn about just one arm. And, since the sampling procedure is to some extent decoupled from the behavior of BASE, one would like to incorporate heuristics to better guide the sampling procedure while retaining a strong regret bound.

We present a novel meta-algorithm, Stochastic Delayed Bandits (SDB) (Algorithm 2), which takes better advantage of stochasticity and heuristics to improve performance in the presence of delay. The basic idea is that in addition to a (possibly stochastic) bandit algorithm BASE, we also take as input an arbitrary (possibly stochastic) algorithm HEURISTIC. SDB tries to follow the probability distribution specified by the heuristic algorithm, but with the constraint that

[1]Lemma 1 in the appendix (available at http://grail.cs.washington.edu/projects/bandit) addresses the case where this assumption is relaxed.

the queues do not get too large. Specifically, we keep an estimate B of τ_{max}, and try to prevent any queue from being larger than B. If a queue for arm i approaches size B, we only allow the algorithm to assign a small amount of probability to arm i, and assign zero probability if it is over B. However, if all queues are small compared to B we allow HEURISTIC to control the sampling distribution. Intuitively, this approach keeps the queues small; this is desirable because if the queues grow too large there could be a large gap between our current performance and that of BASE run online. We prove a regret bound on SDB within a constant factor of that of QPM-D, and in our experiments we will see that it performs much better in practice.

Algorithm 1 General Queue-Based Bandit Algorithm

```
 1: Let $\alpha \in (0,1]$ be a user-defined mixing weight
 2: Create an empty FIFO queue $Q[i]$ for $i \in \{1,\dots,N\}$
 3: Initialize queue size counters $S_i = 0$ for $i \in \{1,\dots,N\}$
 4: Initialize $B = 1$; Initialize list $L$ as an empty list
 5: Let $p$ be BASE's first arm-pull prob. distribution
 6: Let $h$ be HEURISTIC's first arm-pull prob. distribution
 7: Draw $I \sim p$
 8: for $t = 1$ to $\infty$ do
 9:     while $Q[I]$ is not empty do
10:         Remove reward $r$ from $Q[I]$, decrement $S_I$
11:         Update BASE with $r$ and $I$
12:         Get BASE arm distribution $p$; Draw $I \sim p$
13:     $q = $ GETSAMPLINGDIST(...)
14:     Sample from environment with distribution $q$
15:       (once if online updates or else for one batch)
16:     Increment $S_i$ by the # of times arm $i$ was sampled
17:     for each sample that has arrived do
18:         Observe reward $r_i$ and add to $Q[i]$.
19:         Add the delay (number of timesteps since
20:           selected) of this sample to $L$
21:         Update HEURISTIC with reward $r_i$
22:         Get next HEURISTIC arm distribution $h$
23:     Set $B$ to the maximum delay in $L$.
```

Theorem 1. *For algorithm 1 with any choice of procedure* GETSAMPLINGDIST *and any online bandit algorithm* BASE, $\mathbb{E}[R_T] \le \mathbb{E}\left[R_T^{\text{BASE}}\right] + \sum_{i=1}^{N} \Delta_i \mathbb{E}[S_{i,T}]$ *where $S_{i,T}$ is the number of elements pulled for arm i by time T, but not yet shown to* BASE.

The proof of Theorem 1 is provided in the appendix (available at http://grail.cs.washington.edu/projects/bandit). The proof sketch is that for each item, it has either been assigned to queue or it has been consumed by BASE. We know the regret of BASE on the samples it pulls from the queues is upper bounded by $\mathbb{E}\left[R_T^{\text{BASE}}\right]$, so combining that with the regret of the items in each queue gives us the stated regret.

Theorem 2. *For SDB,* $\mathbb{E}[R_T] \le \mathbb{E}\left[R_T^{\text{BASE}}\right] + N\tau_{max}$ *in the online updating setting, and* $\mathbb{E}[R_T] \le \mathbb{E}\left[R_T^{\text{BASE}}\right] + 2N\tau_{max}$ *in the batch updating setting.*

The proof of Theorem 2 is in the appendix (available at http://grail.cs.washington.edu/projects/bandit). The proof

Algorithm 2 Stochastic Delayed Bandits (SDB) uses GETSAMPLINGDISTSDB in line 13 of Algorithm 1

```
 1: procedure GETSAMPLINGDISTSDB(h, p, I, S, B, $\alpha$)
 2:     $\mathcal{A} = \{1,\dots,N\} - I$  ▷ $\mathcal{A}$ are the arms we're willing to alter
 3:     $q = h$  ▷ $q$ will be a modification of $h$ that avoids sampling from full queues.
 4:     $q = (1-\alpha)q + \alpha p$  ▷ We start by mixing in a small amount of $p$ to so that if the queues $h$ wants to sample are full we are close to $p$.
 5:     for all $i$ do
 6:         $u_i = \max(\frac{B-S_i}{B}, 0)$  ▷ $u_i$ is the maximum allowable probability for $q_i$
 7:     while $\exists i \in \mathcal{A}$ such that $q_i > u_i$ do
 8:         $d = q_i - u_i$  ▷ $d$ is the probability mass to redistribute to other arms
 9:         $q_i = u_i$; $\mathcal{A} = \mathcal{A} - i$  ▷ Set $q_i$ to $u_i$ and do not allow $q_i$ to change further
10:         $sum = \sum_{j \in \mathcal{A}} q_j$
11:         for all $j \in \mathcal{A}$ do
12:             $q_j = q_j \times \frac{sum+d}{sum}$  ▷ Redistribution of probability lost to unmodified arms
13:     return $q$
14: end procedure
```

proceeds by showing that the size of any queue cannot exceed τ_{max} given online updating, and $2\tau_{max}$ given batch updating . In the online update case the proof is simple because we always assign 0 probability to arms with more than B elements, and $B \le \tau_{max}$ always. In the batch update case the proof proceeds similarly, except it can take up to one batch to notice that the queue has gotten too large and zero out the probabilities, so we incur an extra additive term of τ_{max}.

Heuristics When choosing a sampling distribution, SDB starts with a weighted combination of the distribution produced by the HEURISTIC and BASE algorithms, where $\alpha \in (0,1]$ of the probability mass is put on BASE.[2] It is important to observe that SDB's regret bound (Theorem 2) depends on neither α's value nor HEURISTIC's behavior.[3] This allows us to set α very low, thereby primarily trusting an arbitrary heuristic algorithm (at least initially), while achieving the best known theoretical guarantees in the online updating delay case, and within a constant factor in the batch updating case. So in the presence of delay, SDB allows arbitrary heuristics to help bandit algorithms get better reward in favorable cases, while avoiding poor worst-case behavior, as we shall see in the empirical results.

Incorporating Prior Data SDB can be used to handle access to prior data by initially placing the prior dataset in the queues and setting B to M, the maximum size of any queue. SDB ensures that the algorithm does not perform

[2]α cannot be 0, since empty queue $Q[I]$ might have zero probability, resulting in a division by 0 on line 12 of algorithm 2.

[3]This is because a bad heuristic will quickly fill up the queues for the bad arms, causing us to ignore it and instead follow the arms selected by BASE.

poorly when handed a (possibly adversarially) skewed distribution of arms, instead making sure that it remains close to its online performance. This is because if τ_{max} is taken to be the max of the true maximum delay and M, the same guarantees apply, since initially $S_i < \tau_{max}$ and $B < \tau_{max}$, therefore the induction in the proof of Theorem 2 holds. In the experiments we will see that certain popular bandit algorithms can perform poorly when handed all prior data, motivating our approach.

Efficient Unbiased Bandit Evaluation

Deploying a good bandit algorithm typically involves some degree of parameter tuning and comparisons between different approaches (e.g. index-based vs Bayesian) (Vermorel and Mohri 2005; Chapelle and Li 2011). Deployment in the real world can be too expensive, and building accurate simulators is challenging, especially when interacting with people. A preferable approach is unbiased offline evaluation using a previous dataset. The state-of-the-art in unbiased offline evaluation of nonstationary bandit algorithms is the rejection sampling based replayer of Li et al. 2011 developed for the more challenging case of contextual bandits.

However, our queue method allows us to be much more data efficient in the special case of context-free bandits: Li's replayer will "reject" many samples, but our approach stores them in queues for later use. Furthermore, we will show that our method has an unbiasedness guarantee and does not require knowledge of the distribution from which the data was collected. Our evaluator proceeds similarly to Algorithm 1, but all queues are initialized with samples from a past dataset D. When the algorithm samples an empty queue after T updates (line 13), we terminate and return an estimate of the algorithm's performance at every timestep up to time T.

Traditional unbiasedness can be defined as, for all t:

$$\mathbb{E}\left[\hat{r_t}\right] = \sum_{s'=\{i,\ldots,j\}\in S_t} p(s'|\theta)\mu_j, \tag{1}$$

where $\hat{r_t}$ is the estimate of reward at time t, S_t is the list of all possible arm sequences (i.e. possible histories of arm pulls up to time t), $p(s'|\theta)$ is the probability that given the true environment our algorithm will select sequence s' at time t, and μ_j is the true mean of arm j (the last arm in sequence s'). Both our and Li's algorithm satisfy this unbiasedness property, given an infinite dataset. In practice, however, we will only have a finite dataset; for some sequences neither algorithm will be able to output an estimate for time t due to lack of data, so that neither estimator satisfies (1).

For our queue replay estimator, however, we can claim a slightly weaker conditional unbiasedness property for the set of sequences $U_t \subseteq S_t$ for which we can return an estimate before hitting the end of a queue. Specifically, we can claim that $\mathbb{E}\left[r_t | s \in U_t\right] = \sum_{s'=\{i,\ldots,j\}\in U_t} p(s'|s' \in U_t, \theta)\mu_j$.

Theorem 3. *Queue Replay Evaluation estimates are unbiased conditioned on the fact that the bandit algorithm produces a sequence of actions for which we issue an estimate. Specifically,* $\mathbb{E}\left[r_t | s \in U_t\right] = \sum_{s'=\{i,\ldots,j\}\in U_t} p(s'|s' \in U_t, \theta)\mu_j$.

The proof can be found in the appendix (available at http://grail.cs.washington.edu/projects/bandit) and builds upon Lemma 4 from Joulani et al. 2013.

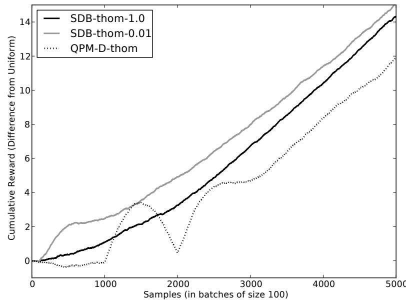
The obvious issue this raises is that if our algorithm puts considerable probability mass on sequences not in U_t, having the correct expectation for sequences in U_t is meaningless. However, after one run of our algorithm, we can observe when we hit the end of a queue and only report estimates for earlier timesteps. If we want to estimate the reward over multiple randomizations of our algorithm up to some time T, if our evaluation frequently terminates earlier than T steps we can reduce T such that the probability of being over T is sufficiently high (perhaps over a certain threshold, such as 95% or 99%). This will give us a degree of confidence in the reported estimates since rewards are bounded in [0, 1], so a very small percentage of episodes cannot have a large effect on the average reward. Furthermore, note that the order in which we draw items from the queue does not impact the theoretical properties since each item is iid, so we can re-randomize between runs of the algorithm to get a better estimate of general performance.

Simulations

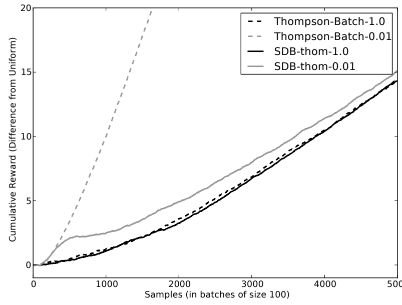
In this section we compare the simulated performance of our algorithms. To give a better picture of performance, simulations are averaged over many randomly-chosen but similar environments. The environments in Figures 1a, 1b, 1d, and 1e consist of 10 Gaussian arms with means picked uniformly at random in the interval [0.475, 0.525] and variances picked in the interval [0, 0.05]. The environment in Figure 1f is also 10 Gaussian arms with means picked in the interval [0.9, 1.0] and variances in [0, 0.05]. In both cases the Gaussians were clamped to [0, 1] to make them fit the standard bandit formulation. Finally, Figure 1c has two Bernoulli arms, with means picked uniformly from [0.475, 0.525]. These environments were kept the same between candidate algorithms to reduce the variance. For each simulation, each candidate algorithm was run once on 1000 different environments, and subtracted from a uniform policy (to ensure an accurate baseline, the uniform policy was run 10 times in each of those 1000 environments).

Since SDB takes black-box algorithms as input, we have complete freedom in choosing bandit algorithms from the literature. We focus on Thompson Sampling (Thompson 1933), a Bayesian method that has gained enormous interest in the bandit community recently, since it typically outperforms most other algorithms in practice (Chapelle and Li 2011) while possessing optimal regret bounds (Agrawal and Goyal 2013) in the online case. Our rewards are non-binary and so we use the Gaussian-Gaussian variant presented in Agrawal et al. 2013[4]. The distribution over arms is easy to sample from but challenging to compute exactly; thus we sample 100 times to construct an approximate distribution from which to sample. We also present some results using UCB (Auer, Cesa-Bianchi, and Fischer 2002), a popular

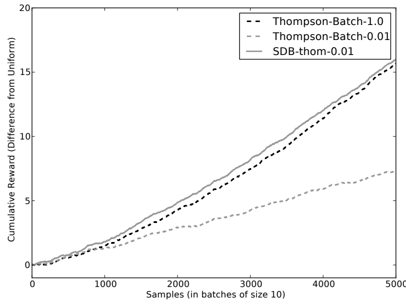
[4]Note that this does not mean we assume the arm reward distributions are Gaussian, Agrawal's method has optimal bounds given *any* distribution.



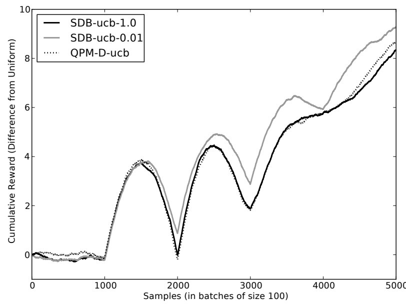
(a) Comparing to QPM-D with Thompson Sampling as the black-box algorithm.



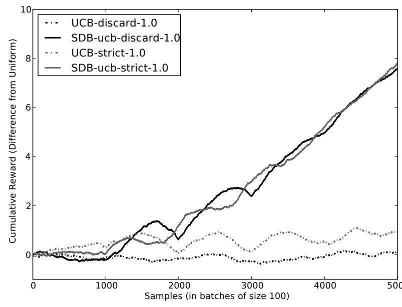
(b) Comparing to heuristics with Thompson Sampling as the black-box algorithm.



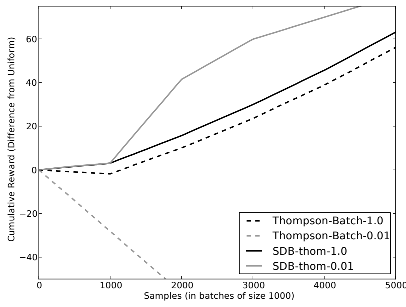
(c) An example where the heuristic Thompson-Batch-0.01 performs worse.



(d) Comparing to QPM-D using UCB as the black-box algorithm.



(e) Two example algorithms that perform poorly when handed all samples but well inside SDB.



(f) An example of a case where handing a prior dataset from a poor arm hurts Thompson-Batch but not SDB.

Figure 1: Simulation Results. SDB-thom-X refers to running SDB with Thompson-1.0 as the BASE algorithm and Thompson-X as the HEURISTIC algorithm, and likewise for UCB.

bandit algorithm that also has optimal guarantees in the online case. The SDB parameter α which controls the weighting between HEURISTIC and BASE is always set to 0.01 in our simulations and experiments.

Here we focus on the batch updating model of delay with batches of various sizes. For results in the online updating case (in which case our regret-bound exactly matches the best-known), see the appendix (available at http://grail.cs.washington.edu/projects/bandit).

Figure 1a shows a comparison of SDB to QPM-D using Thompson Sampling algorithms, in environments with batches of size 100. The parameter in all of our graphs refers to variance scaling: 1.0 provides optimal online theoretical guarantees, while lower values cause Thompson Sampling to overexploit in the worst case. For SDB-thom-X, the BASE algorithm is always Thompson-1.0 to provide guarantees, while the HEURISTIC algorithm is Thompson-X. QPM-D also uses Thompson-1.0 as BASE. The results show that QPM-D does poorly initially, being equivalent to uniform for the first 1000 samples (since it devotes one batch to each of the 10 arms), and also performs poorly long-term. The strange shape of QPM-D is due to its determinism: when it samples a bad arm it must devote an entire batch to it, resulting in periods of poor performance. With an added heuristic, SDB uses it to substantially improve performance, especially early on, while retaining theoretical guarantees.

Figure 1b shows a comparison of SDB to heuristic approaches. Thompson-Batch-X refers to an algorithm which hands Thompson-X all samples after each batch instead of using queues (good regret bounds are not known for this approach). SDB-thom-1.0 and Thompson-Batch-1.0 perform similarly, which is encouraging since SDB has been shown to have strong theoretical guarantees while Thompson-Batch has not. SDB does look worse than Thompson-Batch-0.01 since it explores more to retain theoretical guarantees.

However, ignoring theoretical guarantees, why not just use a heuristic like the Thompson-Batch-0.01 algorithm, since it performs much better than the other algorithms in Figure 1b? To explain why, see Figure 1c. In this environment we see that Thompson-Batch-0.01 performs very poorly compared to Thompson-Batch-1.0, since the arm means are so hard to distinguish that it tends to exploit bad arms. However, SDB-0.01 performs as good or even a little better than Thompson-Batch-1.0, showing that it avoids being misled in environments where the heuristic is bad.

Figure 1d is similar to Figure 1a except we have used UCB instead of Thompson. Here we don't see a difference between SDB-ucb-1.0 and QPM-D, since there is no stochasticity in the black-box algorithms for SDB to exploit. However, SDB still sees improvement when heuristics are added, while retaining the theoretical guarantees.

Of course, UCB and Thompson are not the only reasonable bandit algorithms. In Figure 1e we have illustrated the benefit of SDB with two examples of bandit algorithms that perform very poorly if they are run with delay, but perform well under SDB. UCB-Strict acts similarly to UCB, but defaults to random performance if it observes a reward from an arm that does not have the highest upper confidence interval. UCB-Discard is also based on UCB but discards rewards it observes from arms that do not have the highest confidence interval (an alternative approach to ensuring batch and online performance are similar). As expected, handing all samples from a batch to these algorithms hurts performance, but running them inside SDB preserves good performance. Although many bandit algorithms do not exhibit this extreme behavior, these examples show that queues are needed to guarantee good performance for black-box algorithms.

Figure 1f shows an example of Thompson-Batch performing badly given prior data. In this case we supplied one batch (1000 samples) of prior data from the worst arm. Thompson-Batch puts slightly more weight on that arm initially, resulting in poor initial performance. The effect is exacerbated when decreasing the tuning parameter, as Thompson-Batch-0.01 focuses entirely on exploiting the poor arm, causing very poor performance. SDB, however, chooses not to sample that arm at all for the first batch since it already has enough data, resulting in better performance in this case. It also makes use of the heuristic to improve performance, avoiding the poor performance of the heuristic used alone.[5]

Experiments

Experiment Setup Treefrog Treasure (Figure 2a) is an educational fractions game developed by the Center for Game Science. Players control a frog and jump off walls, occasionally needing to jump through correct point on a numberline. We want to select optimal settings for numberlines to promote student learning and engagement, which we model as a standard bandit problem. Our experiment has 9 numberline parameters, e.g. whether or not ticks are shown to help guide the player. Given a fixed first set of lines, we want to either increase the difficulty level of one parameter (e.g. removing tick marks) or stay at the same difficulty, corresponding to $9 + 1 = 10$ bandit arms. Since the game randomly increases the difficulty of each parameter as time goes on, the choice of arm affects not only the immediate next line but also future lines. The data collection procedure assigned each player to an arm using a fixed (almost uniform) distribution. Our dataset was collected from BrainPOP.com, an educational website focused on school-aged children. It consisted of 4,396 students, each of whom formed a single (arm, reward) pair.

Our reward is a mix of engagement and learning. At the beginning of the game we assign a randomized in-game pretest with 4 numberlines, and include only those players in the experiment who score at most 2 out of 4. Then, we randomly pick a number of numberlines the player must complete before we give a 4-question in-game posttest. The posttest questions are randomly drawn from the same distribution as the pretest, which allows us to define a noisy metric of learning ranging from -2 to 4 based on the difference between pretest and posttest scores. The reward cannot be less than -2 since students must have gotten at least two lines incorrect on the pretest to be included. Finally, if the player quits before the posttest they are given -0.98 reward. We transform rewards to [0,1] before running our algorithms to satisfy their assumptions.

Experiment Results First, given real data and various bandit algorithms, how does our replayer compare to Li's rejection sampling estimator (Li et al. 2011)? If the data collection policy were unknown, it is unclear how to apply Li's approach since it requires knowing the distribution from which the data was drawn. However in our case we do know this distribution, so both approaches produce unbiased sequences of interactions with the environment. Hence the key criteria is how many pulls each approach can generate estimates for: longer sequences give us a better sense of the long-term performance of our algorithms.

Figure 2b compares data efficiency between our queue-based estimator and the rejection sampling based replayer of (Li et al. 2011) on our 4,396 person dataset when running various algorithms without delay. We see that our performance is only slightly better given a stochastic policy (Thompson), since rejection sampling leverages the revealed randomness to accept many samples. However, with deterministic policies such as UCB, our queue-based estimator can evaluate policies for much longer. This limitation of rejection sampling-based replayers was noted in (Dudík et al. 2012) as an open problem, although it is unclear whether our estimator can be adapted to the contextual bandits setting.

To generate the cumulative reward graph we average each algorithm over 1000 samples from our SDB-based replayer (as in the simulations, Uniform received 10,000 samples). To ensure reliability, all results had at least 99.5% of trajectories reach the graphed length before hitting an empty queue. We plot rewards in the original range, where 1 unit corresponds to 1 number line improvement on the posttest.

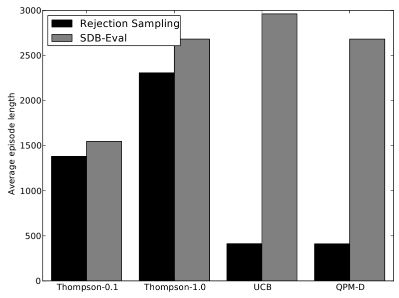
Figure 2c shows results on our data with batch size 100. We see that SDB exploits the heuristic to perform much better than simply using Thompson-1.0. The other algorithm which maintains strong theoretical guarantees, QPM-D, does much worse empirically. We are encouraged at the good early performance of SDB, which is critical in settings such as educational games where experiments may be terminated or players may leave if early performance is poor.

Related Work

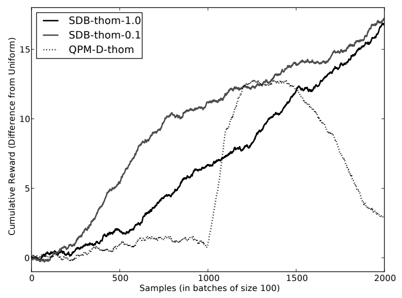
Bandits with delay In this work we build off of Joulani's 2013 approach to dealing with delay in stochastic multi-armed bandits, specifically their QPM-D algorithm. We improve over this work by (a) better exploiting stochastic algorithms to improve performance, as we show in our experiments, (b) explicitly considering the batch update setting, (c) incorporating an arbitrary heuristic to improve per-

[5]Although SDB robustly handles a single batch of prior data, a very large prior dataset consisting of many batches might negatively impact both theoretical guarantees and performance. Addressing this issue is left for future work.

(a) Treefrog Treasure: players guide a frog through a physics-based world, solving number line problems.



(b) The queue-based estimator outperforms the rejection-sampling replayer for deterministic policies.



(c) Results (using our queue-based estimator) on educational game data. SDB does better by leveraging the heuristic.

Figure 2: Experimental Results.

formance while retaining strong theoretical guarantees, (d) showing how queue-based methods can be used to incorporate arbitrary prior datasets and (e) showing how queue-based methods can be used to evaluate algorithms offline in an unbiased and highly data-efficient manner, without needing to know the original sampling distribution.

There have been prior approaches to handling delay in bandits prior to Joulani. Dudik et al. 2011 assumed the delay was a fixed, known constant and online updating was allowed. Desautels et al. 2012 analyzed the case of Gaussian Process Bandits, and developed an algorithm which relied on online updating with worse regret bounds than Joulani et al. 2013. Recent work in lock-in bandits (Komiyama, Sato, and Nakagawa 2013) bears similarity to the batch update formulation but does not allow stochastic policies, and proves worse regret bounds than Joulani et al. 2013.

Adding heuristics Although we are not aware of any work explicitly addressing balancing a heuristic and a principled algorithm in a bandit framework, this problem bears similarities to expert advice problems. While the traditional experts setting assumes full information (not bandit feedback) (Littlestone and Warmuth 1994), the multi-armed bandits with expert advice problem is a better fit. However, as McMahan et al. 2009 notes, the theoretical guarantees on solutions to this problem do not hold if the experts learn based on which arms we pull. McMahan explains that without restrictive assumptions on how the experts learn, achieving good regret in this case becomes an extremely challenging reinforcement learning problem. Our approach is considerably simpler, while retaining strong theoretical guarantees and achieving good empirical performance.

Evaluating bandits offline The standard approach to offline bandit evaluation is to build a simulator, either from scratch (e.g. (Auer, Cesa-Bianchi, and Fischer 2002)) or using collected data (e.g. section 4 of (Chapelle and Li 2011)). However, building a good simulator is challenging and comes without strong theoretical guarantees, sparking interest in data-driven estimators. Initial investigations ((Langford, Strehl, and Wortman 2008), (Strehl et al. 2010)) focused on the problem of evaluating stationary policies, however we are interested in nonstationary policies that learn over time. Although there has been work in constructing biased evaluators of nonstationary policies that tend to perform well in practice ((Nicol, Mary, and Preux 2014) and (Dudík et al. 2012)), the state of the art in unbiased estimation (without some other good reward estimator) continues to be the rejection-sampling based replayer proposed by Li et al. in (Li et al. 2011) and (Dudík et al. 2012), which requires a known sampling distribution. We compare to this estimator and show that not only we can provide an unbiasedness guarantee without knowing the sampling distribution, but our evaluator is considerably more data-efficient.

Conclusion

In this paper we identified the queue method, and showed how it can be used to solve four problems more effectively than prior solutions. Specifically, we presented Stochastic Delayed Bandits (SDB), an algorithm which leverages the distributions of black-box stochastic bandit algorithms to perform well in the face of delay. Although typically using an arbitrary heuristic worsens or eliminates theoretical guarantees, we showed how SDB can take advantage of a heuristic to improve performance while retaining a regret bound close to the best-known in the face of delay. We also showed how SDB is more robust to prior data than typical approaches. Finally, we presented a conditionally unbiased estimator which uses the queue method to be highly data-efficient even without knowing the distribution from which the data was collected. These approaches have strong theoretical guarantees, and have good performance when evaluated both on synthetic simulations and on a real-world educational games dataset. Future work includes better methods for incorporating very large prior datasets, exploring how the queue method can be applied to more complex settings, and developing stochastic algorithms that match the best-known regret bound in the batch updating case.

Acknowledgements

This work was supported by the NSF Graduate Research Fellowship grant No. DGE-0718124, the Office of Naval Research grant N00014-12-C-0158, the Bill and Melinda

Gates Foundation grant OPP1031488, the Hewlett Foundation grant 2012-8161, Adobe, Google, and Microsoft.

References

Agrawal, S., and Goyal, N. 2013. Further optimal regret bounds for thompson sampling. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, 99–107.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Chapelle, O., and Li, L. 2011. An empirical evaluation of thompson sampling. In *NIPS*, 2249–2257.

Desautels, T.; Krause, A.; and Burdick, J. 2012. Parallelizing exploration-exploitation tradeoffs with gaussian process bandit optimization. *arXiv preprint arXiv:1206.6402*.

Dudik, M.; Hsu, D.; Kale, S.; Karampatziakis, N.; Langford, J.; Reyzin, L.; and Zhang, T. 2011. Efficient optimal learning for contextual bandits. *arXiv preprint arXiv:1106.2369*.

Dudík, M.; Erhan, D.; Langford, J.; and Li, L. 2012. Sample-efficient nonstationary policy evaluation for contextual bandits. *arXiv preprint arXiv:1210.4862*.

Joulani, P.; Gyorgy, A.; and Szepesvari, C. 2013. Online learning under delayed feedback. In *Proceedings of The 30th International Conference on Machine Learning*, 1453–1461.

Komiyama, J.; Sato, I.; and Nakagawa, H. 2013. Multi-armed bandit problem with lock-up periods. In *Asian Conference on Machine Learning*, 116–132.

Langford, J.; Strehl, A.; and Wortman, J. 2008. Exploration scavenging. In *Proceedings of the 25th international conference on Machine learning*, 528–535. ACM.

Li, L.; Chu, W.; Langford, J.; and Wang, X. 2011. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the fourth ACM international conference on Web search and data mining*, 297–306. ACM.

Littlestone, N., and Warmuth, M. K. 1994. The weighted majority algorithm. *Information and computation* 108(2):212–261.

McMahan, H. B., and Streeter, M. J. 2009. Tighter bounds for multi-armed bandits with expert advice. In *COLT*.

Nicol, O.; Mary, J.; and Preux, P. 2014. Improving offline evaluation of contextual bandit algorithms via bootstrapping techniques. *arXiv preprint arXiv:1405.3536*.

Scott, S. L. 2010. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry* 26(6):639–658.

Strehl, A. L.; Langford, J.; Li, L.; and Kakade, S. 2010. Learning from logged implicit exploration data. In *NIPS*, volume 10, 2217–2225.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 285–294.

Vermorel, J., and Mohri, M. 2005. Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML 2005*. Springer. 437–448.