PARAMETRIZED STOCHASTIC MULTI-ARMED BANDITS WITH BINARY REWARDS

BY

CHONG JIANG

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Professor Rayadurgam Srikant

ABSTRACT

In this thesis, we consider the problem of multi-armed bandits with a large number of correlated arms. We assume that the arms have Bernoulli distributed rewards, independent across arms and across time, where the probabilities of success are parametrized by known attribute vectors for each arm, as well as an unknown preference vector. For this model, we seek an algorithm with a total regret that is sub-linear in time and independent of the number of arms. We present such an algorithm, which we call the Three-phase Algorithm, and analyze its performance. We show an upper bound on the total regret which applies uniformly in time. The asymptotics of this bound show that for any $f \in \omega(\log(T))$, the total regret can be made to be $O(f(T))$, independent of the number of arms.

To my parents, for their love and support

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1
1.1 Motivation 1
1.2 Model 2
1.3 Prior Work 3

CHAPTER 2 ALGORITHM AND MAIN RESULTS 5
2.1 Definitions 5
2.2 Three-Phase Algorithm 5
2.3 Main Results 8
2.4 Generalizations of the Basic Model 14

CHAPTER 3 SIMULATIONS 16
3.1 Simulation Examples 17
3.2 Discussion 18

CHAPTER 4 CONCLUSIONS 22

APPENDIX A SIMULATION CODE 23

REFERENCES 28

CHAPTER 1

INTRODUCTION

1.1 Motivation

The stochastic multi-armed bandit problem is the following: suppose we are allowed to choose to "pull," or play, any one of m slot machines (also known as one-armed bandits) in each of T timesteps, where each slot machine generates a reward according to its own distribution which is unknown to us. Furthermore, the rewards are independent across machines, and independent and identically distributed across time slots. The choice of which arm to pull may be a function of the sequence of past pulls and the sequence of past rewards. If our goal is to maximize the total reward obtained, taking expectation over the randomness of the outcomes, ideally we would pull the arm with the largest mean at all times. However, we do not know in advance which arm has the largest mean, so a certain amount of exploration is required. Too much exploration, though, wastes time that could be spent reaping the reward offered by the best arm. This exemplifies the fundamental trade-off between exploration and exploitation present in a wide class of online machine learning problems.

We consider a model for multi-armed bandit problems in which a large number of arms are present, where the expected rewards of the arms are coupled through an unknown parameter of lower dimension. That is, the reward distributions between the arms are no longer independent. In this way, it is no longer necessary for each arm to be investigated in order to estimate the expected reward from that arm. Instead, we can estimate the underlying parameter; in this way, each pull can yield information about multiple arms. We present a simple algorithm, and a bound on the expected regret for any time horizon under this algorithm. While possibly sub-optimal, this algorithm has an asymptotic expected total regret which can be bounded above by functions of time which are independent of all problem parameters, including the number of arms.

This model is applicable to certain e-commerce applications: suppose an online retailer has a large number of related products, and wishes to maximize revenue or profit coming from a certain set of customers. If the preferences of this set of customers are known, the list of items which are displayed can be sorted in descending order of expected revenue or profit. However, we may not know a priori what this preference vector is, so we wish to learn online by sequentially presenting each user with an item, observing whether the user buys the item, and then updating an internal estimate of the preference vector. In practice, it could be impractical to track each individual user's preferences; partitioning the set of customers into demographic groups and estimating each group's preference vector may suffice.

As a concrete example, imagine an online camera store, with hundreds of different camera models in stock. However, there are perhaps closer to ten features which people will compare when deciding which, if any, to purchase. There are permanent features of the camera itself, such as megapixel count, brand name, and year of introduction, as well as extrinsic features, such as price, review scores, and item popularity. All of these features might be considered by the customer in order to decide whether or not to buy the camera. If bought, the store gains a profit corresponding to the item. A key distinction of our model, when compared to previous work, is the incorporation of this inherently binary choice customers are faced with: to buy or not to buy.

Another similar example is an online search engine, in which having the results ordered by relevancy increases user satisfaction, as well as ad revenue from sponsored links. While the user has some real-valued probability of clicking any given link, the primary piece of feedback the search engine gets is whether or not the link has been clicked, again a binary decision.

1.2 Model

Our model consists of a multi-armed bandit with m arms (items) and n underlying parameters (attributes), where $m \geq n$, and potentially $m \gg n$. Each arm i is associated with a constant n-dimensional attribute vector u_i, and we assume that $\text{rank}\,[u_1, \ldots, u_m] = n$. There is also a constant but unknown n-dimensional preference vector $z^* \in \mathbb{R}^n$. The quality $\beta_i = u_i^T z^*$ of arm i is a scalar indicating how desirable the item is to a user. The expected reward of an arm i assuming a given z is defined as $\alpha_i(z) = f\left(u_i^T z\right) = \dfrac{1}{1 + \exp\left(-u_i^T z\right)}, \;\; \forall\, i \in \{1, \ldots, m\}$;

thus we see that the expected rewards of all of the arms are coupled through z^*. We note that our results are applicable to more general functions f; we will comment more on this later. For notational simplicity, let $\alpha_i^* = \alpha_i(z^*)$. Let $b \in \{1, \ldots, m-1\}$ denote the number of equally best arms, so that $\alpha_1^* = \alpha_2^* = \cdots = \alpha_b^* > \alpha_{b+1}^* \geq \cdots \geq \alpha_m^*$. At each timestep t up to a finite time horizon T, a policy will choose to pull exactly one arm, call this arm C_t, and a reward X_t will be obtained, where $X_t \sim \text{Ber}(\alpha_{C_t}^*)$. We wish to find policies g which maximize the total expected reward, $\sum_{t=1}^{T} X_t$, or equivalently, minimize the expected total regret, $E_g\left[\sum_{t=1}^{T}(\alpha_1^* - X_t)\right] = T \cdot \alpha_1^* - E_g[\sum_{t=1}^{T} \alpha_{C_t}^*]$.

1.3 Prior Work

For an introduction and survey of classical multi-armed bandit problems and their variations, see Mahajan and Teneketzis [1]. One of the earliest breakthroughs on the classical multi-armed bandit problem came from Gittins and Jones [2], who showed that the optimal policy assigns an index to each arm, now known as the Gittins index, and pulls the arm with the largest Gittins index. Other proofs of this optimality have been given later by Weber [3] and Tsitsiklis [4]. Whittle [5] proved a similar index-based result in the "restless bandit" variation of this model, in which the arms which are not pulled also evolve in time. While these policies greatly simplify a single m-dimensional problem into m 1-dimensional problems, it is still, in general, too computationally complex for online learning.

Lai and Robbins [6] proved an achievable $O(\log T)$ lower bound for the expected total regret of the stochastic multi-armed bandit problem in the case of independent arms, and they termed algorithms which achieved the bound to be asymptotically efficient. However, the pre-constant in the $O(\log T)$, in general, scales linearly with the number of arms. Agrawal *et al.* [7,8] considered a similar model with a different parametrization of the arms, and obtained asymptotically efficient policies in both the i.i.d. and Markov cases. Anantharam *et al.* [9,10] considered another similar model with "multiple plays," in which multiple arms are played at each timestep, and also obtained optimal policies in both the i.i.d. and Markov cases. Agrawal *et al.* [11] showed that the Lai and Robbins model, amended with the addition of a "switching cost" to discourage frequent switching of chosen arms, still has asymptotically efficient policies.

Mersereau *et al.* [12] proposed a model in which the expected rewards are affine functions of a single scalar parameter z, which has some underlying continuous distribution Z. Furthermore, the set of arms is allowed to be a bounded, convex region in $\mathbb{R}^n$, in which case m is uncountably infinite. They then derived a policy whose expected total regret is $\Theta\left(\sqrt{T}\right)$ and whose Bayes risk, the expectation of the regret over Z, is $\Theta\left(\log T\right)$. Rusmevichientong and Tsitsiklis [13] expanded this model to allow for a multi-variate parameter z of dimension n, and showed that the expected total regret (ignoring $\log T$ factors) is $\Theta\left(n\sqrt{T}\right)$, and that the Bayes risk is $\Theta\left(n\sqrt{T}\right)$. Dani *et al.* [14] independently considered a nearly identical model, and obtained similar results.

Auer *et al.* [15] were the first to consider a non-stochastic version of the multi-armed bandit problem, in which the rewards are no longer drawn from an unknown distribution, but can instead be adversarially generated. The resultant total weak regret, calculated by comparison with the single arm which is best over the entire time horizon, is shown to be $O\left(\sqrt{mT}\right)$. The change from logarithmic to polynomial regret in this model is due to having rewards which are time-dependent and potentially adversarially generated, instead of being drawn from a time-independent distribution.

Audibert *et al.* [16] considered the problem of best arm identification in a stochastic multi-armed bandit setting, but where the goal is to maximize the probability of determining the best arm at the end of a time horizon, as opposed to the usual goal of minimizing total regret over a time horizon. This model is useful when considering exploration and exploitation as occurring in series, instead of in parallel. The authors' algorithm successively rejects the worst arm, based on empirical mean, until only the best arm remains. The probability of error is shown to be upper bounded by a decaying exponential in T.

Auer *et al.* [17] investigated the finite-time regret of the multi-armed bandit problem, assuming bounded but otherwise arbitrary reward distributions. Using upper confidence bound algorithms, where the confidence interval of an arm shrinks as the arm is subjected to more plays, they achieve a logarithmic upper bound on the regret, uniform over time, that scales with the "gaps" between the expected rewards for the arms.

A common idea used in crafting policies to solve the multi-armed bandit problem is that of the doubling trick [18,19]. This technique is used to convert a parametrized algorithm which works on a time horizon T, along with its corresponding bound, into a non-parametrized algorithm that runs forever, with an upper bound that holds uniformly over time.

CHAPTER 2

ALGORITHM AND MAIN RESULTS

2.1 Definitions

Let $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$, and $\mathbb{N}_1 = \{1, 2, \ldots\}$.

Let $f, g : \mathbb{N}_0 \to \mathbb{R}$. We write

$$f(t) \in O\left(g(t)\right) \text{ if } \exists c > 0 \text{ s.t. } \lim_{t\to\infty} \frac{f(t)}{g(t)} \leq c,$$
$$f(t) \in o\left(g(t)\right) \text{ if } \forall c > 0, \lim_{t\to\infty} \frac{f(t)}{g(t)} < c,$$
$$f(t) \in \omega\left(g(t)\right) \text{ if } \forall c > 0, \lim_{t\to\infty} \frac{f(t)}{g(t)} > c.$$

Let $\log^*(x)$, the iterated logarithm, be defined by

$$\log^*(x) = \begin{cases} 0, & \text{if } x \leq 1 \\ 1 + \log^*\left(\log x\right), & \text{if } x > 1 \end{cases}.$$

2.2 Three-Phase Algorithm

We first present an algorithmic description of a policy for the multi-armed bandit problem described in Section 1.2. This algorithm, which we call the Three-phase Algorithm, will depend on a scheduling function $g : \mathbb{N}_1 \to \mathbb{N}_0$, such that g is strictly increasing, and that

$g(l) \in o\left(\exp\left(k \cdot l\right)\right),\ \forall k > 0$. Since g is not surjective in general, its inverse g^{-1} is not defined over all of $\mathbb{N}_0$; however, the strict monotonicity of g allows us to define g^{-1} in the following natural way: let $g^{-1}(t) = \max\left\{1, \max\left\{l \in \mathbb{N}_1 : g(l) \leq t\right\}\right\},\ \forall t \in \mathbb{N}_0$. In Theorem 1, we will show that the expected total regret of this policy is $E[R_T] \in O\left(g^{-1}(T)\right)$, with a pre-constant which is independent of the problem parameters n, m, $\{u_i\}_{i=1}^m$, and z^*.

Algorithm 1 Three-phase Algorithm

Require: $g : \mathbb{N}_1 \to \mathbb{N}_0$, such that g is strictly increasing, and $g(l) \in o\left(\exp\left(k \cdot l\right)\right),\ \forall k > 0$
Require: $\Sigma = \{\sigma(i)\}_{i=1}^n \subseteq \{1, \ldots, m\}$, such that $U_\Sigma = \left[u_{\sigma(1)}, \ldots, u_{\sigma(n)}\right]$ has rank n.
$t \leftarrow 1$, $l \leftarrow 1$
$q_{i,0} \leftarrow 0$, $q_{i,1} \leftarrow 0\ \forall i \in \Sigma$
while $t \leq T$ and $\exists i \in \Sigma, j \in \{0,1\}$, such that $q_{i,j} = 0$ **do**
  Pull arm $C_t \leftarrow \min\left\{i \in \Sigma : q_{i,0} = 0 \text{ or } q_{i,1} = 0\right\}$ and obtain reward X_t {Phase 0}
  $q_{C_t,X_t} \leftarrow q_{C_t,X_t} + 1$
  $t \leftarrow t + 1$
end while
loop
  Pull arm $C_t \leftarrow \sigma\left((l-1) \pmod n + 1\right)$ and obtain reward X_t {Phase 1}
  $q_{C_t,X_t} \leftarrow q_{C_t,X_t} + 1$
  $t \leftarrow t + 1$
  Form the estimates $\hat{\alpha}_i \leftarrow \dfrac{q_{i,1}}{q_{i,0} + q_{i,1}},\ \forall i \in \Sigma$
  Form the estimate $\hat{z} \leftarrow \left(U_\Sigma^T\right)^{-1} \begin{bmatrix} f^{-1}\left(\hat{\alpha}_{\sigma(1)}\right) \\ \vdots \\ f^{-1}\left(\hat{\alpha}_{\sigma(n)}\right) \end{bmatrix}$
  $C_{(l)} \leftarrow \arg\max_{i \in \{1,\ldots,m\}} \alpha_i(\hat{z})$
  for $s \leftarrow 1$ **to** $g(l)$ **do**
    Pull arm $C_t \leftarrow C_{(l)}$ and obtain reward X_t {Phase 2}
    $t \leftarrow t + 1$
  end for
  $l \leftarrow l + 1$
end loop

The algorithm requires a selection of n arms, $\Sigma = \{\sigma(i)\}_{i=1}^n \subseteq \{1, \ldots, m\}$, such that $U_\Sigma = \left[u_{\sigma(1)}, \ldots, u_{\sigma(n)}\right]$ has rank n. Such a choice exists since we assume $[u_1, u_2, \ldots, u_m]$ has rank n. The algorithm starts by pulling arms in Σ until each has yielded both a 1 and a 0. Note this takes a random, but a.s. finite number of timesteps (together called Phase 0). After this, the algorithm proceeds in epochs. Epoch l consists of a single exploration pull (called Phase 1) of an arm in Σ, the estimation of the underlying parameter z^*, and $g(l)$ exploitation pulls (called Phase 2) of the best arm given that estimate. If we impose a time horizon of T, epochs $1, 2, \ldots$ are appended until the time horizon T has been reached. The

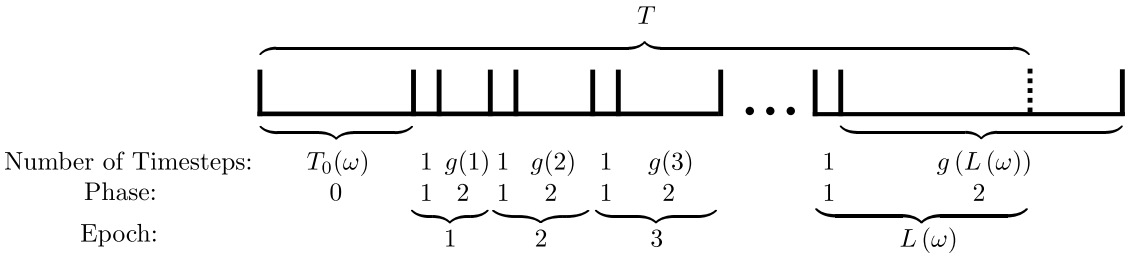

Figure 2.1: Given a time horizon T, we partition the T timesteps into Phase 0, Phase 1, and Phase 2 timesteps. The Phase 1 and Phase 2 timesteps are grouped into a total of $L(\omega)$ epochs.

three phases are illustrated in Figure 2.1.

For each timestep t which is either Phase 0 or Phase 1, which implies an arm $i \in \Sigma$ was chosen, we increment the empirical count q_{i,X_t} by 1. Prior to each Phase 2 timestep during epoch l, there have already been l Phase 1 pulls. We can form empirical estimates for α_i^* based on only the Phase 0 and Phase 1 timesteps, namely $\hat{\alpha}_{i,l} = \dfrac{q_{i,1}}{q_{i,0} + q_{i,1}}$, $\forall i \in \Sigma$. Note that being in Phase 2 implies we have completed Phase 0, which ensures that $q_{i,0} \geq 1$ and $q_{i,1} \geq 1$, and thus $0 < \hat{\alpha}_{i,l} < 1$.

Since f is strictly increasing and continuous, its inverse exists. Since U_Σ is an $n \times n$ matrix with full rank, $\left(U_\Sigma^T\right)^{-1}$ exists. We can now form an estimate for z^*, namely

$$\hat{z} = \left(U_\Sigma^T\right)^{-1} \begin{bmatrix} f^{-1}\left(\hat{\alpha}_{\sigma(1)}\right) \\ \vdots \\ f^{-1}\left(\hat{\alpha}_{\sigma(n)}\right) \end{bmatrix}$$

and choose $C_t = \arg\max_{i \in \{1,\ldots,m\}} \alpha_i(\hat{z})$.

Remark 2.2.1 *In practice, LU decomposition, instead of matrix inversion, can be used to solve for $\hat{z}$. Also, since f is strictly increasing, the estimated best arm in epoch l, $C_{(l)}$, can be computed as* $\arg\max_{i \in \{1,\ldots,m\}} \left(u_i^T \hat{z}\right)$.

We shall point out some of the ideas behind this algorithm. First, the algorithm is defined to run indefinitely; to obtain the total regret for any finite time horizon T, we simply terminate the algorithm when timestep T has been reached. This achieves the same outcome as an application of the doubling trick, in that the algorithm is not dependent on a time horizon

T. Our algorithm is similar to the algorithm UCB2 of [17]. The main difference is that in our exploration phases, the choice of arm exploits the correlation model that we have assumed in our problem. Furthermore, as we will see later, unlike UCB2, the lengths of the exploitation phases are chosen to grow sub-exponentially in the epoch number in order to obtain a regret bound that grows (slightly larger than) logarithmically in the time horizon. As we gain more information and are able to estimate z^* more accurately, we can spend a greater fraction of timesteps exploiting the arm we think is best; this is achieved by choosing a suitable scheduling function g to control the ratio of the number of exploitation (Phase 2) pulls versus exploration (Phase 1) pulls, as a function of the epoch number l.

2.3 Main Results

Note that there is only randomness in $\{X_t\}_{t=1}^T$, since the Three-phase Algorithm is otherwise deterministic. We will use ω to denote the sample-paths of $\{X_t\}_{t=1}^T$. Let $L(\omega)$ denote the number of Phase 1 timesteps up until time T, or equivalently, the number of epochs (including partial epochs, as the final one may be truncated), for a given sample-path ω.

Let $R_{i,T}(\omega)$ be the total regret up to timestep T in the Phase i timesteps for a sample-path ω. Now, let $R_T(\omega) = R_{0,T}(\omega) + R_{1,T}(\omega) + R_{2,T}(\omega)$, the total regret up to timestep T for a sample-path ω. Our goal is to find an upper bound on $E[R_T]$, the expected total regret. In particular, we are interested in the asymptotic behavior of the upper bound as $T \to \infty$.

Lemma 2.3.1 *For the Three-phase Algorithm, we have the following bound on the expected total Phase 0 regret up to timestep T:*

$$E[R_{0,T}] \leq \alpha_1^* \sum_{i \in \Sigma} \left[\frac{1}{\alpha_i^*(1-\alpha_i^*)} \right].$$

Proof:

Note that $E[R_{0,T}] \leq \alpha_1^* E\left[\sum_{i \in \Sigma} W_i\right]$, where $W_i \sim \text{Geo}(\alpha_i^*) + \text{Geo}(1-\alpha_i^*)$ is the time it

takes to first observe a 1 and subsequently observe a 0 from an arm $i \in \Sigma$. Thus,

$$\begin{aligned} E\left[R_{0,T}\right] \leq & \alpha_1^* \sum_{i \in \Sigma} \left[\left(\frac{1}{\alpha_i^*} + \frac{1}{1-\alpha_i^*} \right) \right] \\ = & \alpha_1^* \sum_{i \in \Sigma} \left[\frac{1}{\alpha_i^* \left(1-\alpha_i^*\right)} \right]. \end{aligned}$$

□

Lemma 2.3.2 *For the Three-phase Algorithm, we have the following bound on the expected total Phase 1 regret up to timestep T:*

$$E\left[R_{1,T}\right] \leq \left(\alpha_1^* - \frac{1}{n} \sum_{i \in \Sigma} \alpha_i^* \right) \cdot \left(E[L] + n\right).$$

Proof:

$$\begin{aligned} E\left[R_{1,T}\right] = & E\left[\sum_{l=1}^{L(\omega)} \left[\alpha_1^* - \alpha^*_{\sigma((l-1)(\mathrm{mod}\ n)+1)} \right] \right] \\ \leq & \sum_{l=1}^{E\left[\left\lfloor \frac{L(\omega)}{n} \right\rfloor + 1\right]} \left(\alpha_1^* n - \sum_{i \in \Sigma} \alpha_i^* \right) \\ \leq & \left(\alpha_1^* - \frac{1}{n} \sum_{i \in \Sigma} \alpha_i^* \right) \cdot \left(E[L] + n\right). \end{aligned}$$

□

Lemma 2.3.3 *For the Three-phase Algorithm, for a given choice of scheduling function g, we have the following bound on the expected total Phase 2 regret up to timestep T:*

$$E\left[R_{2,T}\right] \leq \alpha_1^* \sum_{l=1}^{L'-1} \left[2n \exp\left(-\frac{l}{n} \cdot \gamma \right) g(l) \right] + \frac{\alpha_1^*}{2} E[L],$$

where L' is a constant which depends on n, $\{u_i\}_{i=1}^m$, and z^*.

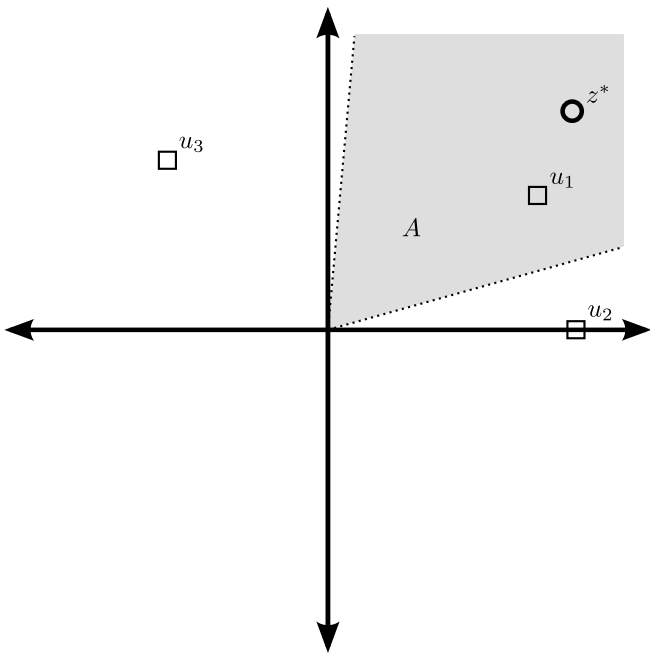

Figure 2.2: As an example, consider a scenario with $n = 2$ and $m = 3$. The arms $\{u_i\}_{i=1}^3$ and the preference vector z^* are located at the indicated points. The shaded region is A.

Proof: Recall that $\alpha_i^* = f\left(u_i^T z^*\right)$, where $f(\beta) = \dfrac{1}{1+\exp(-\beta)}$ is strictly increasing and continuous. Thus f^{-1} is well defined, strictly increasing and continuous. Since $\alpha_1^* = \alpha_2^* = \cdots = \alpha_b^* > \alpha_{b+1}^* \geq \cdots \geq \alpha_m^*$, and because $f\left(u_i^T z\right)$ is continuous in z and defined over $\mathbb{R}^n$, it follows that there exists a neighborhood of z^*, denoted A, such that $\arg\max_{i\in\{1,\ldots,m\}} \alpha_i(z) \in \{1,\ldots,b\},\ \forall z \in A$. Since U_Σ is full rank, A must contain an open parallelotope centered at z^*, $B_{z^*}(\delta) = \left\{z : \left\|U_\Sigma^T z - U_\Sigma^T z^*\right\|_\infty < \delta\right\}$, where $\delta > 0$ and is largest possible. An example of the problem parameters and the induced region A is shown in Figure 2.2.

Consider any $z \in B_{z^*}(\delta)$. By definition, $|u_i^T z - u_i^T z^*| < \delta,\ \forall i \in \Sigma$. This is equivalent to $|f^{-1}(\alpha_i(z)) - f^{-1}(\alpha_i^*)| < \delta,\ \forall i \in \Sigma$. Since f^{-1} is continuous, this is equivalent to having a set of constants $\{\underline{\alpha}_i, \overline{\alpha}_i\}_{i\in\Sigma}$, such that $\underline{\alpha}_i < \alpha_i(z) < \overline{\alpha}_i$, where $\underline{\alpha}_i = f(f^{-1}(\alpha_i^*) - \delta)$ and $\overline{\alpha}_i = f(f^{-1}(\alpha_i^*) + \delta),\ \forall i \in \Sigma$.

For a Phase 2 timestep during epoch l, the algorithm forms the empirical average rewards $\hat{\alpha}_i,\ \forall i \in \Sigma$. If it is the case that $\underline{\alpha}_i < \hat{\alpha}_i < \overline{\alpha}_i,\ \forall i \in \Sigma$, then by the discussion above, $\hat{z} \in B_{z^*}(\delta) \subseteq A$, and hence, $C_t = \arg\max_{i\in\{1,\ldots,m\}}\{u_i^T \hat{z}\} \in \{1,\ldots,b\}$, and we will have chosen one of the best arms, accumulating zero regret.

Note that during epoch l, we have that $q_{i,0}+q_{i,1} \geq 2+\left\lfloor \frac{l}{n} \right\rfloor \geq \frac{l}{n},\ \forall i \in \Sigma$, where the first term is due to the Phase 0 pulls and the second term is due to the Phase 1 pulls. Furthermore, during epoch l, $\hat{\alpha}_i$ is a sum of $q_{i,0}+q_{i,1}$ i.i.d. Ber (α_i^*) random variables, $\forall i \in \Sigma$. By the Chernoff bound,

$$P\left(\hat{\alpha}_i < \underline{\alpha}_i\right) \leq \exp\left[-\left(q_{i,0}+q_{i,1}\right) \cdot D\left(\underline{\alpha}_i || \alpha_i^*\right)\right] \leq \exp\left[-\frac{l}{n} \cdot D\left(\underline{\alpha}_i || \alpha_i^*\right)\right], \text{ and}$$
$$P\left(\hat{\alpha}_i > \overline{\alpha}_i\right) \leq \exp\left[-\left(q_{i,0}+q_{i,1}\right) \cdot D\left(\overline{\alpha}_i || \alpha_i^*\right)\right] \leq \exp\left[-\frac{l}{n} \cdot D\left(\overline{\alpha}_i || \alpha_i^*\right)\right],\ \forall i \in \Sigma,$$

where $D\left(p||q\right) = p \cdot \log \frac{p}{q} + (1-p) \cdot \log \frac{1-p}{1-q}$ is the K-L divergence between two Bernoulli distributions.

Let $\gamma = \min_{i \in \Sigma} \min\left\{D\left(\underline{\alpha}_i || \alpha_i^*\right), D\left(\overline{\alpha}_i || \alpha_i^*\right)\right\}$. Note that from the definitions of $\underline{\alpha}_i$ and $\overline{\alpha}_i$, it follows that $\underline{\alpha}_i < \alpha_i^* < \overline{\alpha}_i,\ \forall i \in \Sigma$. Since $D\left(p||q\right) = 0 \iff p = q$, we have that $\gamma > 0$.

By the union bound, $P\left(\exists i \in \Sigma : \hat{\alpha}_i \notin \left(\underline{\alpha}_i, \overline{\alpha}_i\right)\right) \leq 2n \exp\left(-\frac{l}{n} \cdot \gamma\right)$.

Reviewing the chain of implications, we have

$$\begin{aligned}
P\left(\hat{z} \notin A\right) \leq & P\left(\hat{z} \notin B_{z^*}(\delta)\right) \\
= & P\left(\left\|U_\Sigma^T \hat{z} - U_\Sigma^T z^*\right\|_\infty > \delta\right) \\
= & P\left(\exists i \in \Sigma : \left|u_i^T z - u_i^T z^*\right| > \delta\right) \\
= & P\left(\exists i \in \Sigma : \left|f^{-1}\left(\hat{\alpha}_i\right) - f^{-1}\left(\alpha_i^*\right)\right| > \delta\right) \\
= & P\left(\exists i \in \Sigma : \hat{\alpha}_i \notin \left(\underline{\alpha}_i, \overline{\alpha}_i\right)\right) \\
\leq & 2n \exp\left(-\frac{l}{n} \cdot \gamma\right).
\end{aligned}$$

Then, we have a bound on the expected per-timestep regret $r_{2,l}$ during epoch l:

$$\begin{aligned}
E\left[r_{2,l}\right] = & E\left[r_{2,l} | \hat{z} \in A\right] \cdot P\left(\hat{z} \in A\right) + E\left[r_{2,l} | \hat{z} \notin A\right] \cdot P\left(\hat{z} \notin A\right) \\
\leq & 0 \cdot P\left(\hat{z} \in A\right) + \alpha_1^* \cdot P\left(\hat{z} \notin A\right) \\
\leq & 2\alpha_1^* n \exp\left(-\frac{l}{n} \cdot \gamma\right).
\end{aligned}$$

We can now find the expected total regret in the phase 2 times up to time T:

$$\begin{aligned}
E\left[R_{2,T}\right] =& E\left[\sum_{l=1}^{L(\omega)} r_{2,l}\cdot g(l)\right] \\
=& \sum_{l=1}^{E[L]}\left[E\left[r_{2,l}\right]\cdot g(l)\right] \\
\leq& \sum_{l=1}^{E[L]}\left[2\alpha_1^* n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right] \\
=& \sum_{l=1}^{L'-1}\left[2\alpha_1^* n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right]+\sum_{l=L'}^{E[L]}\left[2\alpha_1^* n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right] \\
\leq& \alpha_1^*\sum_{l=1}^{L'-1}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right]+\sum_{l=L'}^{E[L]}\frac{\alpha_1^*}{2} \\
\leq& \alpha_1^*\sum_{l=1}^{L'-1}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right]+\frac{\alpha_1^*}{2}E[L]
\end{aligned}$$

where $L'=\max\left\{l : 2n\exp\left(-\frac{l}{n}\gamma\right)g(l)>\frac{1}{2}\right\}$ is a parameter which depends on n, $\{u_i\}_{i=1}^m$, and z^* (and is therefore unknown to the algorithm). However, since we have assumed $g(l)\in o\left(\exp\left(k\cdot l\right)\right)\ \forall k>0$, it follows that $\lim_{l\to\infty}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right]=0$, and thus L' is finite. Therefore, the sum $\sum_{l=1}^{L'-1}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right]$ is well defined.

□

Theorem 2.3.4 *For the Three-phase Algorithm, we have the following asymptotic bound on the expected total regret up to timestep T: $E\left[R_T\right]=O\left(g^{-1}(T)\right)$.*

Proof:

Let us break down the total time T as $T=T_0(\omega)+T_1(\omega)+T_2(\omega)$, where $T_i(\omega)$ is the number of timesteps in Phase i for sample-path ω. Note that for all sample-paths ω in which $L\left(\omega\right)\geq 2$, we have that $g\left(L\left(\omega\right)-1\right)\leq T$, where the left side counts the number of Phase 2 timesteps in the penultimate epoch. Thus, $L\left(\omega\right)\leq g^{-1}(T)+1$, and hence, $E[L]\leq g^{-1}(T)+1$.

Using Lemmas 2.3.1, 2.3.2, and 2.3.3,

$$E[R_T] = E[R_{0,T} + R_{1,T} + R_{2,T}] \leq K + \left(\frac{3}{2}\alpha_1^* - \frac{1}{n}\sum_{i\in\Sigma}\alpha_i^*\right) g^{-1}(T) \in O\left(g^{-1}(T)\right),$$

where

$$\begin{aligned} K =& \alpha_1^* \sum_{i\in\Sigma}\left[\frac{1}{\alpha_i^*(1-\alpha_i^*)}\right] + \left(\alpha_1^* - \frac{1}{n}\sum_{i\in\Sigma}\alpha_i^*\right)\cdot(n+1) \\ &+ \alpha_1^* \sum_{l=1}^{L'-1}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right] + \frac{\alpha_1^*}{2} \end{aligned}$$

is a constant which depends on n, $\{u_i\}_{i=1}^m$, and z^* (and is therefore unknown to the algorithm), but is finite for any valid set of problem parameters. Note that K does not depend on T, and is therefore $O(1)$; thus the asymptotic expected total regret is $O\left(g^{-1}(T)\right)$.

□

Corollary 2.3.5 *If $g^{-1}(t) \in \omega(\log(t))$, then g is a valid scheduling function for the Three-phase Algorithm.*

Proof: Since $g^{-1}(t) \in \omega(\log(t))$, by definition, $\lim_{t\to\infty} \frac{g^{-1}(t)}{k\cdot\log(t)} > 1$, $\forall k > 0$. Equivalently, $\lim_{t\to\infty} \frac{k_1\cdot g^{-1}(t)}{\log(t)} > \frac{1}{k_2}$, $\forall k_1, k_2 > 0$.

Since $g : \mathbb{N}_1 \to \mathbb{N}_0$ is strictly increasing, $\lim_{l\to\infty} g(l) = \infty$. Also, note that by construction, $\forall l \in \mathbb{N}_1$, $g^{-1}(g(l)) = l$. Thus, we can make the substitution $t = g(l)$,

$$\lim_{l\to\infty} \frac{k_1\cdot g^{-1}(g(l))}{\log(g(l))} = \lim_{t\to\infty} \frac{k_1\cdot l}{\log(g(l))} = \lim_{t\to\infty} \frac{\exp(k_1\cdot l)}{g(l)} > \frac{1}{k_2},$$

Hence $\lim_{t\to\infty} \frac{g(l)}{\exp(k_1\cdot l)} < k_2$. Thus, $\forall k_1 > 0$, $g(l) \in o(\exp(k_1\cdot l))$, and g is therefore a valid scheduling function, as desired.

□

Corollary 2.3.6 *An expected total regret of* $E[R_T] \in O\left(\log(T) \cdot \log^*(T)\right)$ *is achievable with the Three-phase Algorithm.*

Proof: Pick $g_{LLS}(l) = \max\left\{t \in \mathbb{N}_1 : \log(t) \cdot \log^*(t) \leq l\right\}$. Now, $g_{LLS}^{-1}(t) = \lfloor \log(t) \cdot \log^*(t) \rfloor$, so $\lim_{t\to\infty} \frac{g_{LLS}^{-1}(t)}{\log(t)} = \lim_{t\to\infty} \log^*(t) \to \infty$. Thus, $g_{LLS} \in \omega\left(\log(t)\right)$ is a valid scheduling function for the Three-phase Algorithm, and a expected total regret of $E[R_T] \in O\left(g^{-1}(T)\right) \subseteq O\left(\log(T) \cdot \log^*(T)\right)$ is achievable.

□

Remark 2.3.7 *In accordance with other results, such as [6], we suspect this problem has a lower bound that is asymptotically* $c \cdot (\log(T))$*, where* c *is dependent on the problem parameters* n*,* $\{u_i\}_{i=1}^m$*, and* z^**. If this is the case, then by including the term* $\log^*(T)$*, we are able to obtain an upper bound which is not tight, but within a factor of* $\log^*(T)$*; however, this allows us to gain simplicity, in the sense that our pre-constant is* $\frac{3}{2}$*, independent of the problem parameters.*

2.4 Generalizations of the Basic Model

2.4.1 Arm-Dependent Rewards

Suppose that each arm i has a potentially different value of the reward, so that instead of a $\{0, 1\}$ reward, it has a $\{0, w_i\}$ reward. Furthermore, suppose that $\{w_i\}_{i=1}^m$ is known. Now, $X_t \sim w_{C_t} \cdot \text{Ber}\left(\alpha_{C_t}^*\right)$ instead of $X_t \sim \text{Ber}\left(\alpha_{C_t}^*\right)$. Let the indices of the arms be sorted by decreasing expected reward $w_i\alpha_i^*$.

Then, we can generalize Theorem 2.3.4 with only minor modifications to the proof, namely,

$$E[R_T] = E[R_{0,T} + R_{1,T} + R_{2,T}] \leq K' + \left(\frac{3}{2}w_1\alpha_1^* - \frac{1}{n}\sum_{i\in\Sigma} w_i\alpha_i^*\right) g^{-1}(T) \in O\left(g^{-1}(T)\right),$$

where

$$K' = w_1\alpha_1^* \sum_{i\in\Sigma}\left[\frac{1}{\alpha_i^*\left(1-\alpha_i^*\right)}\right] + \left(w_1\alpha_1^* - \frac{1}{n}\sum_{i\in\Sigma} w_i\alpha_i^*\right)\cdot(n+1) + w_1\alpha_1^* \sum_{l=1}^{L'-1}\left[2n\exp\left(-\frac{l}{n}\cdot\gamma\right)g(l)\right] + \frac{w_1\alpha_1^*}{2}.$$

Thus, Corollary 2.3.6 is unchanged with the addition of arm-dependent rewards.

2.4.2 Generalized Functional Dependency on Quality

If we generalize the definition of the expected reward of an arm i assuming a preference vector z to be $\alpha_i(z) = f_i\left(u_i^T z\right),\ \forall i \in \{1,\ldots,m\}$, with the condition that $f_i(\beta) : \mathbb{R} \to (0,1)$ is strictly increasing and continuous, but otherwise arbitrary, all of the discussion above still holds. The algorithm only needs a slight modification in the formation of the estimate $\hat{z}$, which is now

$$\hat{z} = \left(U_\Sigma^T\right)^{-1}\begin{bmatrix} f_{\sigma(1)}^{-1}\left(\hat{\alpha}_{\sigma(1)}\right) \\ \vdots \\ f_{\sigma(n)}^{-1}\left(\hat{\alpha}_{\sigma(n)}\right)\end{bmatrix}.$$

CHAPTER 3

SIMULATIONS

The Three-phase Algorithm has been implemented in F# 2.0, and several simulations were run in order to observe its empirical behavior. We will use the following set of problem parameters as a nominal experiment:

$$n = 2,\ m = 4,\ f(\beta) = \frac{1}{1+\exp(-\beta)},\ \{u_1, \ldots, u_m\} = \left\{ \begin{array}{cccc} 2 & 1 & 0.7 & 0 \\ 0 & 0 & 0.3 & 1 \end{array} \right\},\ z^* = \{0.5, 0.4\}.$$

The following set of nominal parameters was provided to Algorithm 1, unless noted otherwise:

$T = 10^8,\ g(l) = l^2,\ \Sigma = \{2, 4\}.$

Each plot below shows the empirical average of the total pseudo-regret over 1000 sample-paths ω. That is, for each sample-path ω and each $t \leq T$, we compute the total pseudo-regret $\overline{R}_t(\omega) = \sum_{s=1}^{t} \alpha^*_{C_s(\omega)}$. This is the average of R_t over all sample-paths with the same sequence of chosen arms $\{C_s(\omega)\}_{s=1}^{t}$. Next, we compute an empirical average of the total pseudo-regret, $\hat{R}_t = \frac{1}{1000} \sum_{i=1}^{1000} \overline{R}_t(\omega_i)$. Note that $\overline{R}_t$ and $\hat{R}_t$ are unbiased estimators of $E[R_t]$, when expectation is taken over all sample-paths ω. We plot this instead of expected total regret due to computational limitations.

3.1 Simulation Examples

Dependence on scheduling function g

The Three-phase Algorithm was run on the nominal problem parameters, with the following scheduling functions: $g(l) = l^a$ for each a in $\{1.5, 2, 3\}$, and $g(l) = g_{LLS}(l)$, as defined in Remark 2.3.6. The results are shown in Figure 3.1.

Dependence on gap $\alpha_1^* - \alpha_2^*$

The Three-phase Algorithm was run on the nominal problem parameters, with the nominal set of arms $\{u_i\}_{i=1}^m$, as well as with the nominal $\{u_i\}_{i=1}^m$ with $(0, 2.45)$ inserted as u_2. The gaps $\alpha_1^* - \alpha_2^*$ are 0.109 and 0.004, respectively. The results are shown in Figure 3.2.

Dependence on Σ

The Three-phase Algorithm was run on the nominal problem parameters, with the sets of selected arms $\Sigma = \{2, 4\}$ and $\Sigma = \{3, 4\}$. The corresponding U_Σ are $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 0.7 & 0 \\ 0.3 & 1 \end{pmatrix}$, respectively. The results are shown in Figure 3.3.

Dependence on m

The Three-phase Algorithm was run on the nominal problem parameters, with the nominal set of arms $\{u_i\}_{i=1}^m$, as well as with the $\{u_i\}_{i=1}^m$ when 1000 other arms are inserted uniformly at random from $[0, 1]^2 \cap \{u : u^T z^* < 1\}$. The results are shown in Figure 3.4.

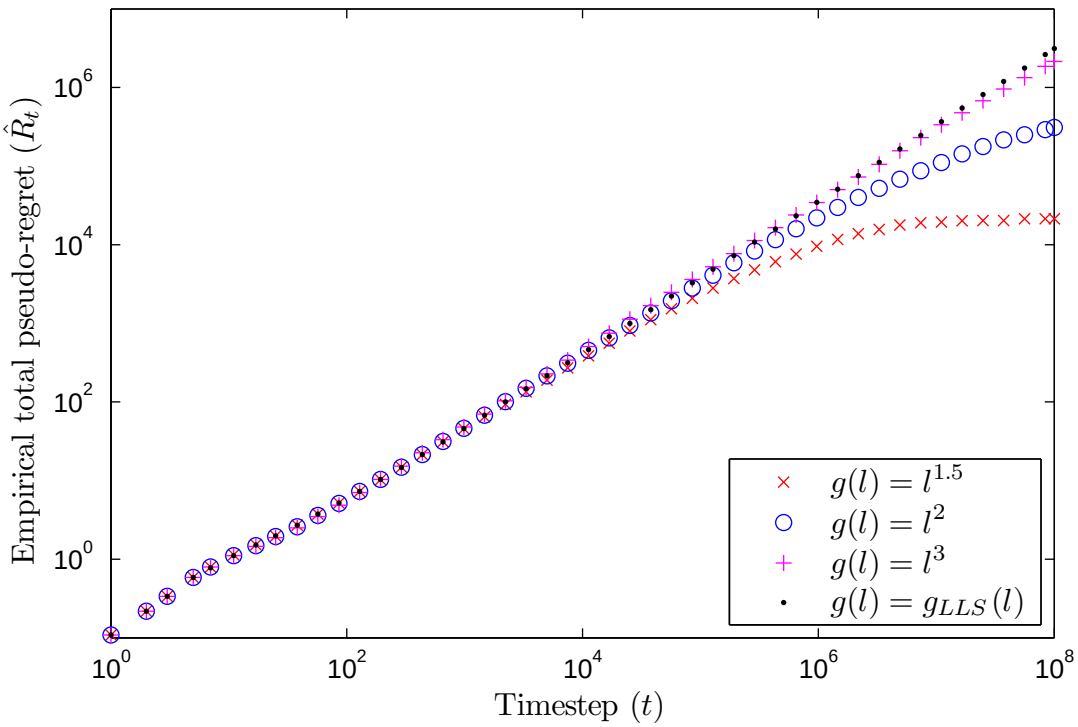

Figure 3.1: Empirical total pseudo-regret vs. timestep, with varying scheduling function g.

3.2 Discussion

In each of the plots, we see the effect of Phase 0, in that the first 80 or so timesteps accumulate linear regret (having slope 1 in our log-log plots), after which there is eventually sub-linear regret (slope < 1 in log-log).

In example 1, we see that the faster the scheduling function g grows, the worse its short time horizon performance. This is readily understandable, as a fast-growing g will take more timesteps to explore and estimate z^* to any particular confidence. Such a scheduling function will accumulate more regret over short time horizons, before becoming superior as $T \to \infty$.

In example 2, we see that a smaller gap $\alpha_1^* - \alpha_2^*$ results in lower regret for short time-horizons. Since our algorithm does not have to explore arms 1 and 2 in order to estimate z^*, the difference arises only in selecting the estimated best arm in Phase 2. With a smaller gap, poor estimations $\hat{z}$ which do not lie in the region A may often cause a nearly-optimal arm to be pulled, incurring only small regret.

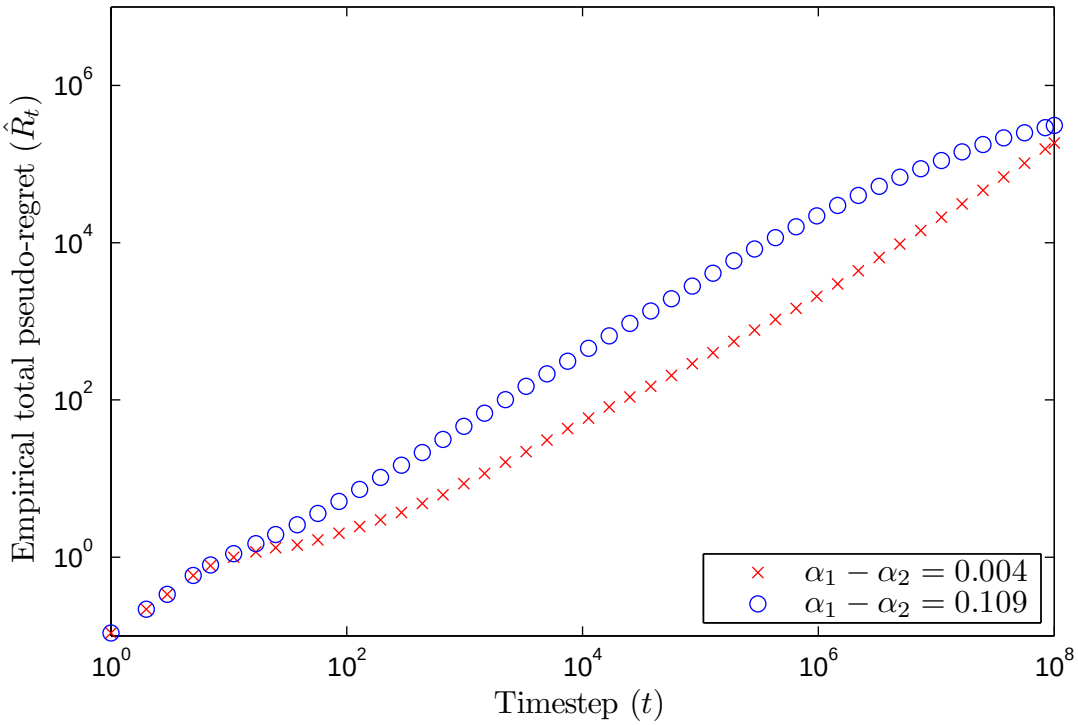

Figure 3.2: Empirical total pseudo-regret vs. timestep, with varying gap $\alpha_1^* - \alpha_2^*$.

In example 3, we see that the choice of Σ affects the resultant regret. This is expected, since our nominal choice of $\Sigma = \{2, 4\}$ yields an orthogonal U_Σ. Another choice of Σ which has a smaller minimum eigenvalue would imply slower estimation of z^* in the direction of the corresponding eigenvector.

In example 4, we see that the addition of sub-optimal arms has little effect on the short time-horizon regret. As only the selection of the estimated best arm in Phase 2 is affected, the dependence of the regret on m is indirectly through introduction of arms which reduce the region A.

These results are in line with our expectations of the algorithm; we see two main factors controlling the rate at which $\hat{z}$ converges to z^*, which in turn controls the short time horizon regret. First, the region $A \subset \mathbb{R}^n$, the set of z for which the best arm is one of the actual best arms assuming z^*, is a crucial property of the problem parameters. For narrow A, it is very difficult to estimate $\hat{z}$ to be within A, and hence we expect to accumulate linear regret in the short time horizon. Such an A occurs when sub-optimal arms are in close proximity (in terms of their attribute vectors $\{u_1, \ldots, u_m\}$) to the best arms. Second, the choice of

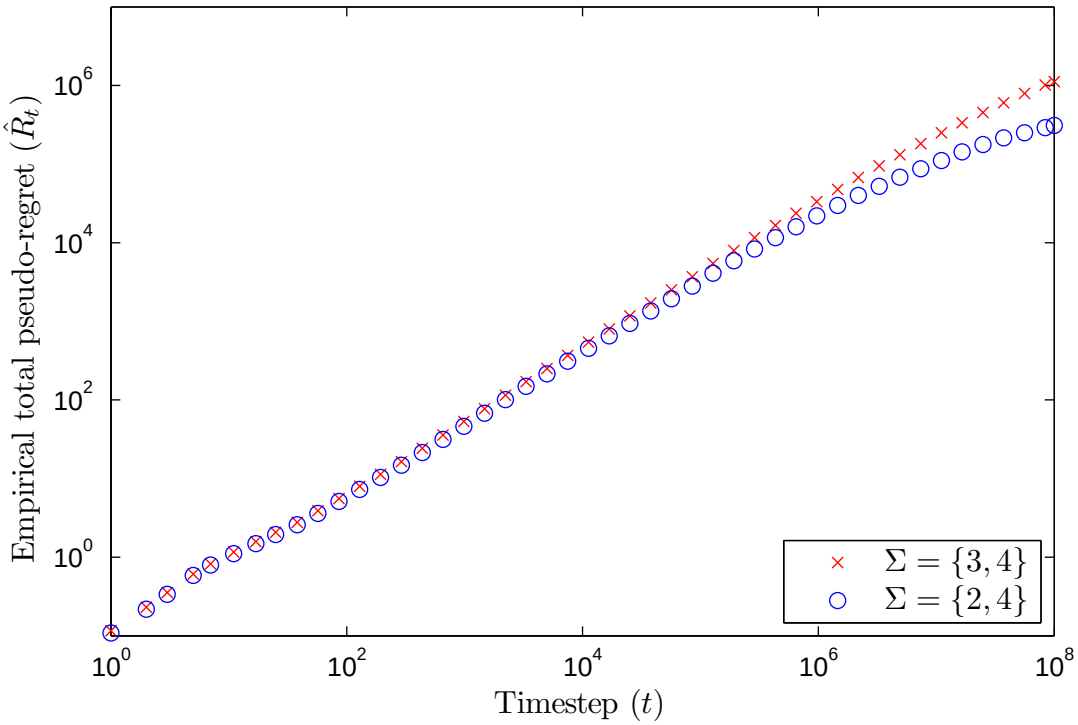

Figure 3.3: Empirical total pseudo-regret vs. timestep, with varying Σ.

Σ, which we provide to the algorithm without knowledge of z^*, also affects the convergence rate of $\hat{z}$. In particular, the components of z^* along the eigenvectors corresponding to small eigenvalues are estimated more slowly. Hence, whenever possible, Σ should be chosen to comprise an orthogonal set of arms.

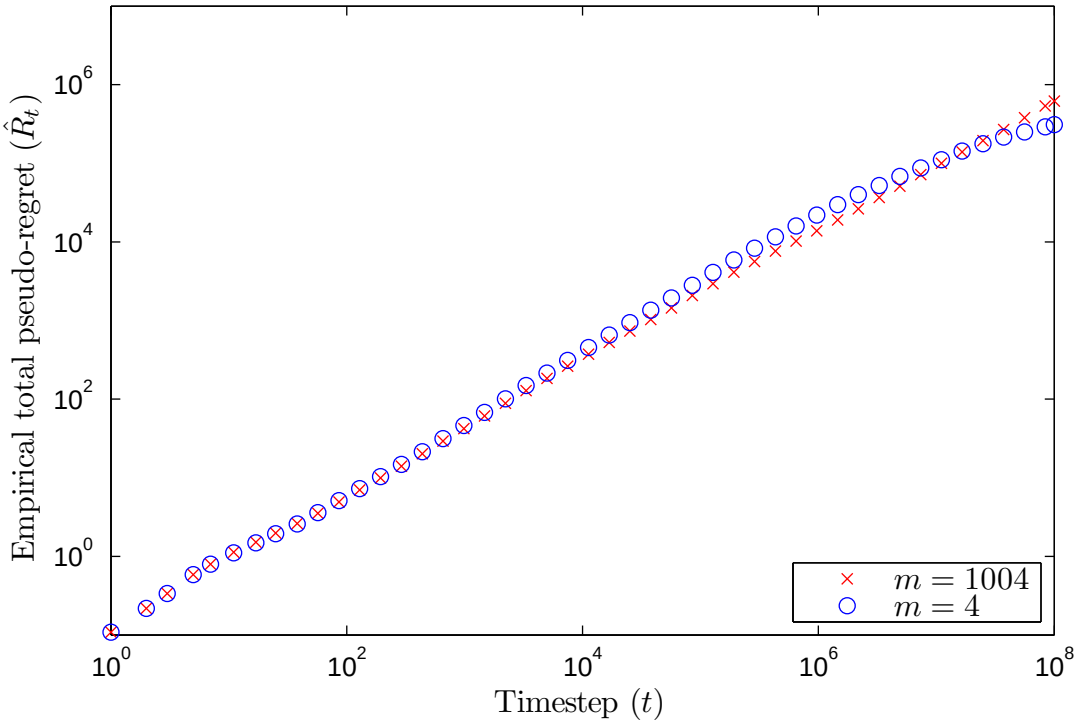

Figure 3.4: Empirical total pseudo-regret vs. timestep, with varying number of arms m.

CHAPTER 4

CONCLUSIONS

We have proposed a class of parametrized multi-armed bandit problems, in which the reward distribution is Bernoulli and independent across arms and across time, with a parameter that is a non-linear function of the scalar quality of an arm. The real-valued quality is an inner product between the unknown preference and known attribute vectors. Under this model, we are able to capture the fundamentally binary choice inherent in certain online machine learning problems. Our proposed algorithm can be implemented efficiently, and is nearly optimal in the sense that its asymptotic expected total regret can be made to be $O\left(g^{-1}(T)\right)$, for any function $g^{-1}(T) \in \omega\left(\log(T)\right)$. This is in contrast to the $O\left(m \log(T)\right)$ bound of Lai and Robbins, and the $O\left(n\sqrt{T}\right)$, large-m bound of Mersereau *et al.*

Several extensions to this work are possible. For example, if z^* is allowed to be time dependent, would it still be possible to estimate z^* in an online fashion, and obtain an asymptotic regret that is close to $\log(T)$? And what if $\{u_i\}_{i=1}^m$ is allowed to be time dependent? Another extension is to multiple plays: If the user pulls d arms at every timestep, with the reward being some function of the d Bernoulli outcomes, how would the asymptotic expected total regret scale with d? Finally, we are also interested in characterizing the short time-horizon performance of the Three-phase Algorithm, through optimization of the Σ and g parameters.

APPENDIX A

SIMULATION CODE

The F# 2.0 code for performing the simulations is included below.

Listing A.1: Algorithm1.fs

```
module Algorithm1

open System
open System.IO
open System.Linq
open Microsoft.FSharp.Collections

// note: T, C, Cl, sigma are 1-indexed
// pX is the pseudo-reward

let printmatrix (m : matrix) =
    let (x, y) = m.Dimensions
    let m2 = m.ToArray2D()
    for i in 0.. x-1 do
        for j in 0 .. y-1 do
            Console.Write(m2.[i,j].ToString() + " ")
        done
        Console.WriteLine()
    done

let printarray (a : 'a []) =
    for (i: int) in 0 .. (a.Length - 1) do
        Console.Write(a.[i].ToString() + " ")
    done
    Console.WriteLine()


let Algorithm1 T a (sigma : int list) f fInv
    (problemParams : int * int * matrix * vector) gFloor (sampleTimeArray : int []) seed =
    //Console.WriteLine("beginning seed: " + seed.ToString())
    let rand = Random seed

    let (n, m, U, z) = problemParams

    let uSigma = Array2D.init n n (fun i j -> U.[i,sigma.[j]-1])
    let uSigmaMNM = MathNet.Numerics.LinearAlgebra.Matrix.Create(uSigma)
    uSigmaMNM.Transpose()
    let uSigmaTransposeLUD = MathNet.Numerics.LinearAlgebra.LUDecomposition(uSigmaMNM)

    // rank check
    if not uSigmaTransposeLUD.IsNonSingular then
        failwith "uSigma cannot be singular"

    let mutable t = 1
    let mutable l = 1
    let q0 = Array.zeroCreate n
    let q1 = Array.zeroCreate n
    let alpha = Array.map f ((U.Transpose * z).ToArray())

    let mutable C = 0
    let mutable X = false
```

```
    let mutable pX = 0.0
    let mutable pXTot = 0.0
    let mutable sampleValueList= []
    let mutable Cl = 0

    let either (x : int Option) (y : int Option) =
        match (x, y) with
          Some x , _       -> Some x
        | _      , Some y -> Some y
        | _      , _       -> None

    let sample_more = Seq.mapi (fun i x -> if 1 > x then Some i else None)

    while ((t <= T) && ((Array.exists ((>) 1) q0) || (Array.exists ((>) 1) q1))) do
        let idx =
            match Seq.tryPick id (Seq.map2 either (sample_more q0) (sample_more q1)) with
                Some id -> id
              | None -> failwith "unreachable"

        C <- sigma.[idx]
        pX <- alpha.[C - 1]
        pXTot <- pXTot + pX
        X <- rand.NextDouble() < pX
        if X then
            q1.[idx] <- q1.[idx] + 1
        else
            q0.[idx] <- q0.[idx] + 1

        if sampleTimeArray.Contains t then
            sampleValueList<- pXTot :: sampleValueList
        t <- t + 1
    done

    while (t <= T) do
        let idx = (l - 1) % n
        C <- sigma.[idx]
        pX <- alpha.[C - 1]
        pXTot <- pXTot + pX
        X <- rand.NextDouble() < pX
        if X then
            q1.[idx] <- q1.[idx] + 1
        else
            q0.[idx] <- q0.[idx] + 1

        if sampleTimeArray.Contains t then
            sampleValueList<- pXTot :: sampleValueList
        t <- t + 1

        let alphaSigmaHat = Array.map2 (fun e0 e1 ->  float(e1) / float(e0 + e1)) q0 q1
        let betaSigmaHat = Array.map fInv alphaSigmaHat
        let betaSigmaHatMNV = MathNet.Numerics.LinearAlgebra.Vector betaSigmaHat
        // betaSigmaHat is a column vector when mapped into a matrix
        let betaSigmaHatMNM = MathNet.Numerics.LinearAlgebra.
                                  Matrix.CreateFromColumns [|betaSigmaHatMNV|]
        let zHatMNM = uSigmaTransposeLUD.Solve(betaSigmaHatMNM)
        let zHat = vector ((zHatMNM.GetColumnVector 0).ToArray())
        let betaHat = (U.Transpose * zHat).ToArray()
        let betaHatMax = Array.max betaHat
        Cl <-  (Array.findIndex ((=) betaHatMax) betaHat) + 1

        let s = Math.Min(gFloor l a, T - t + 1)

        for i in 1 .. s do
            C <- Cl
            pX <- alpha.[C - 1]
            pXTot <- pXTot + pX
            if sampleTimeArray.Contains t then
                sampleValueList<- pXTot :: sampleValueList
            t <- t + 1
        done
        l <- l + 1
    done

    let sampleValueArray = sampleValueList.Reverse().ToArray()
```

```
126         let regretArray = Array.map2 (fun t v ->
                alpha.[0] * float(t) - v) sampleTimeArray sampleValueArray
            regretArray

    let createSampleTimeArray T =
131         let mutable sampleTimeList = []
            let mutable t = 1.0
            let mantissa = 1.5
            while int(t) < T do
                if sampleTimeList.IsEmpty || not (int(t) = sampleTimeList.Head) then
136                 sampleTimeList <- int(t) :: sampleTimeList
                t <- t * mantissa
            done
            if not (T = sampleTimeList.Head) then
                sampleTimeList <- T :: sampleTimeList
141         sampleTimeList.Reverse().ToArray()

    let JSONencode (sampleTimeArray : int []) (regretArray : float []) =
        "{" + "\"t\": [" + String.Join(",", sampleTimeArray) + "], "
            + "\"regret\": [" + String.Join(",", regretArray) + "]}"
146
    let ExportData filename (jsonStr : string) =
        let f = File.CreateText(filename)
        f.WriteLine(jsonStr)
        f.Flush()
151     f.Close()

    let RunExperiment T a sigma f fInv problemParams gFloor numSamples outputFileName =

        let sampleTimeArray = createSampleTimeArray T
156     let regretArrayPSeq = PSeq.init numSamples (fun seed ->
            Algorithm1 T a sigma f fInv problemParams gFloor sampleTimeArray seed)

        let addArray = Array.map2 (+)
        let totalRegretArray = PSeq.reduce addArray regretArrayPSeq
161     let avgRegretArray = Array.map (fun e -> e / float(numSamples)) totalRegretArray

        let jsonStr = JSONencode sampleTimeArray avgRegretArray
        ExportData outputFileName jsonStr
```

Listing A.2: RunAllExperiments.fs

```
 1  module RunAllExperiments

    open System

    let SetupAllExperiments =
 6
        let generateOutputFileName problemParamsName gName numSamples T a sigma =
            // we will always use a T that is a power of 10
            let numZeros = Math.Log10 (float T) |> round
            problemParamsName + "_" + gName + "_" + numSamples.ToString() + "_1e"
11              + numZeros.ToString() + "_" + a.ToString() + "_"
                + String.Join(",", List.map (fun x -> x.ToString()) sigma) + ".txt"

        let mutable experiments = []

16      let canonicalF beta =
            1.0/(1.0 + Math.Exp(-1.0 * beta))

        let canonicalFInv alpha =
            Math.Log (alpha / (1.0 - alpha))
21
        let rec logStar x =
            // all x < 10^1656520 have logStar x <= 4
            match x with
                | x when x <= 1 -> 0
26              | x when x <= 2 -> 1
                | x when x <= 15 -> 2
                | x when x <= 3814279 -> 3
                | _ -> 4

31      let gFloorLogTLogStarTInv x =
```

```
            (Math.Log (float x)) * float(logStar  x)

        let gFloorLogTLogStarT l a =
            let mutable t = 1
            while gFloorLogTLogStarTInv t <= float l do
                t <- t + 1
            done
            t - 1

        let gFloorLPowA l a =
            Math.Pow(float l, a) |> int

        let mutable T = int 1e8
        let mutable a = 2.0
        let mutable sigma = [2; 4]
        let mutable numSamples = 1000
        let f = canonicalF
        let fInv = canonicalFInv
        let mutable outputFileName = "output.txt"

        let problemParamsBasic =
            let n = 2
            let m = 4
            let UTranspose = matrix [ [ 2.0; 0.0];
                                      [ 1.0; 0.0 ];
                                      [ 0.7; 0.3 ];
                                      [ 0.0; 1.0 ] ]
            let U = UTranspose.Transpose
            let z = vector [ 0.5; 0.4 ]
            (n, m, U, z)

        let problemParamsSmallGap =
            let n = 2
            let m = 5
            let UTranspose = matrix [ [ 2.0; 0.0];
                                      [ 0.0; 2.45 ];
                                      [ 1.0; 0.0 ];
                                      [ 0.7; 0.3 ];
                                      [ 0.0; 1.0 ] ]
            let U = UTranspose.Transpose
            let z = vector [ 0.5; 0.4 ]
            (n, m, U, z)

        let problemParamsLargeM =
            let n = 2
            let m = 1004

            let mutable UTranspose = [ [ 2.0; 0.0];
                                       [ 1.0; 0.0 ];
                                       [ 0.7; 0.3 ];
                                       [ 0.0; 1.0 ] ]

            let z = vector [ 0.5 ; 0.4 ]

            let rand = Random 1
            while UTranspose.Length < m do
                let x = rand.NextDouble()
                let y = rand.NextDouble()
                let beta = (vector [x; y]).Transpose * z
                if beta < 1.0 then
                    UTranspose <- [x; y] :: UTranspose
            done
            let sortByBeta = (fun (x : float list) -> -1.0 * ((vector x).Transpose * z) )
            let UTransposeSorted = List.sortBy sortByBeta UTranspose

            // find the positions of [1.0; 0.0] and [0.0; 1.0]
            let betaArray = Seq.map sortByBeta UTranspose |> Array.ofSeq
            let indices = Array.init betaArray.Length id
            System.Array.Sort(betaArray, indices)
            let sigma = [(Array.findIndex ((=) (m-3)) indices) + 1 ;
                         (Array.findIndex ((=) (m-1)) indices) + 1 ]
    //        Console.WriteLine(sigma)
    //        Console.ReadLine() |> ignore
            if not (sigma = [407; 606]) then
```

```
106                 failwith "largeM sigma mismatch"

                let U = (matrix UTransposeSorted).Transpose
                (n, m, U, z)

111         // experiment to look at dependency on choice of a

            // this is the nominal experiment
            a <- 2.0
            outputFileName <- generateOutputFileName "Basic" "l^a" numSamples T a sigma
116         experiments <- (T, a, sigma, f, fInv, problemParamsBasic, gFloorLPowA,
                numSamples, outputFileName) :: experiments

            a <- 1.5
            outputFileName <- generateOutputFileName "Basic" "l^a" numSamples T a sigma
121         experiments <- (T, a, sigma, f, fInv, problemParamsBasic, gFloorLPowA,
                numSamples, outputFileName) :: experiments

            a <- 3.0
            outputFileName <- generateOutputFileName "Basic" "l^a" numSamples T a sigma
126         experiments <- (T, a, sigma, f, fInv, problemParamsBasic, gFloorLPowA,
                numSamples, outputFileName) :: experiments

            // experiment to look at dependency on choice of sigma (cf. nominal)
            a <- 2.0
131         sigma <- [3; 4]
            outputFileName <- generateOutputFileName "Basic" "l^a" numSamples T a sigma
            experiments <- (T, a, sigma, f, fInv, problemParamsBasic, gFloorLPowA,
                numSamples, outputFileName) :: experiments

136         // experiment to look at dependency on gap alpha_1 - alpha_2 (cf. nominal)
            sigma <- [3; 5]
            outputFileName <- generateOutputFileName "SmallGap" "l^a" numSamples T a sigma
            experiments <- (T, a, sigma, f, fInv, problemParamsSmallGap, gFloorLPowA,
                numSamples, outputFileName) :: experiments
141
            // experiment to look at large m (cf. nominal)
            sigma <- [407; 606]
            outputFileName <- generateOutputFileName "LargeM" "l^a" numSamples T a sigma
            experiments <- (T, a, sigma, f, fInv, problemParamsLargeM, gFloorLPowA,
146             numSamples, outputFileName) :: experiments

            // experiment to look at LogTLogStarT scheduling (cf. nominal)
            sigma <- [2; 4]
            outputFileName <- generateOutputFileName "Basic" "LogTLogStarT" numSamples T a sigma
151         experiments <- (T, a, sigma, f, fInv, problemParamsBasic, gFloorLogTLogStarT,
                numSamples, outputFileName) :: experiments

            experiments

156     let RunAllExperiments experiments =
            let output_folder = ""

            for exp in experiments do
                let (T, a, sigma, f, fInv, problemParams, gFloor, numSamples, outputFileName) = exp
161             Console.WriteLine("Beginning experiment " + outputFileName)
                if not (System.IO.File.Exists(output_folder + outputFileName)) then
                    let stopwatch = System.Diagnostics.Stopwatch.StartNew()
                    stopwatch.Start()
                    Algorithm1.RunExperiment T a sigma f fInv problemParams
166                     gFloor numSamples (output_folder + outputFileName)
                    Console.WriteLine("Time taken to run experiment " + outputFileName
                        + ": " + stopwatch.Elapsed.TotalSeconds.ToString() + " seconds")
                else
                    Console.WriteLine("Experiment " + outputFileName + " was skipped")
171         ()

        let experiments = List.toArray SetupAllExperiments
        let p = System.Diagnostics.Process.GetCurrentProcess()
        p.PriorityClass <- System.Diagnostics.ProcessPriorityClass.Idle
176     RunAllExperiments experiments
        Console.WriteLine "All experiments finished"
        let _ = Console.ReadLine()
```

REFERENCES

[1] A. Mahajan and D. Teneketzis, "Multi-armed bandit problems," in *Foundations and Applications of Sensor Management*, A. O. Hero, D. A. Castañón, D. Cochran, and K. Kastella, Eds. Springer-Verlag, 2007, ch. 6, pp. 121–151.

[2] J. C. Gittins and D. M. Jones, "A dynamic allocation index for the sequential design of experiments," *Progress in Statistics*, vol. 1, pp. 241–266, 1974.

[3] R. Weber, "On the Gittins index for multiarmed bandits," *The Annals of Applied Probability*, vol. 2, no. 4, pp. 1024–1033, Nov. 1992.

[4] J. N. Tsitsiklis, "A short proof of the Gittins index theorem," *The Annals of Applied Probability*, vol. 4, no. 1, pp. 194–199, Feb. 1994.

[5] P. Whittle, "Restless bandits: Activity allocation in a changing world," *Journal of Applied Probability*, vol. 25, pp. 287–298, 1988.

[6] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, pp. 4–22, 1985.

[7] R. Agrawal, D. Teneketzis, and V. Anantharam, "Asymptotically efficient adaptive allocation schemes for controlled i.i.d. processes: Finite parameter space," *IEEE Transactions on Automatic Control*, vol. 34, no. 3, pp. 258–267, Mar. 1989.

[8] R. Agrawal, D. Teneketzis, and V. Anantharam, "Asymptotically efficient adaptive allocation schemes for controlled Markov chains: Finite parameter space," *IEEE Transactions on Automatic Control*, vol. 34, no. 12, pp. 1249–1259, Dec. 1989.

[9] V. Anantharam, P. Varaiya, and J. Walrand, "Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays-Part I: I.I.D. rewards," *IEEE Transactions on Automatic Control*, vol. 32, no. 11, pp. 968–976, Nov. 1987.

[10] V. Anantharam, P. Varaiya, and J. Walrand, "Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays-Part II: Markovian rewards," *IEEE Transactions on Automatic Control*, vol. 32, no. 11, pp. 977–982, Nov. 1987.

[11] R. Agrawal, M. Hegde, and D. Teneketzis, "The multi-armed bandit problem with switching cost," in *26th IEEE Conference on Decision and Control, 1987*, vol. 26, Dec. 1987, pp. 1106–1108.

[12] A. J. Mersereau, P. Rusmevichientong, and J. N. Tsitsiklis, "A structured multiarmed bandit problem and the greedy policy," *IEEE Transactions on Automatic Control*, vol. 54, no. 12, pp. 2787–2802, Dec. 2009.

[13] P. Rusmevichientong and J. N. Tsitsiklis, "Linearly parameterized bandits," *Mathematics of Operations Research*, vol. 35, no. 2, pp. 395–411, May 2010.

[14] V. Dani, T. P. Hayes, and S. M. Kakade, "Stochastic linear optimization under bandit feedback," in *Proc. of the 21st Annual Conference on Learning Theory*, Helsinki, Finland, July 2008, pp. 363–374.

[15] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multi-armed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[16] J.-Y. Audibert, S. Bubeck, and R. Munos, "Best arm identification in multi-armed bandits," in *Proc. of the 23rd Annual Conference on Learning Theory*, Haifa, Israel, June 2010, pp. 41–53.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[18] G. Stoltz, "Incomplete information and internal regret in prediction of individual sequences," Ph.D. dissertation, University of Paris-Sud, Nov. 2005. [Online]. Available: http://eprints.pascal-network.org/archive/00001692/

[19] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. New York, NY: Cambridge University Press, 2006.