# A Tutorial on Thompson Sampling

Daniel J. Russo[1], Benjamin Van Roy[2], Abbas Kazerouni[2], Ian Osband[3], and Zheng Wen[4]

[1]Columbia University
[2]Stanford University
[3]Google Deepmind
[4]Adobe Research

November 21, 2017

**Abstract**

Thompson sampling is an algorithm for online decision problems where actions are taken sequentially in a manner that must balance between exploiting what is known to maximize immediate performance and investing to accumulate new information that may improve future performance. The algorithm addresses a broad range of problems in a computationally efficient manner and is therefore enjoying wide use. This tutorial covers the algorithm and its application, illustrating concepts through a range of examples, including Bernoulli bandit problems, shortest path problems, product assortment, recommendation, active learning with neural networks, and reinforcement learning in Markov decision processes. Most of these problems involve complex information structures, where information revealed by taking an action informs beliefs about other actions. We will also discuss when and why Thompson sampling is or is not effective and relations to alternative algorithms.

## 1 Introduction

The multi-armed bandit problem has been the subject of decades of intense study in statistics, operations research, electrical engineering, computer science, and economics. A "one-armed bandit" is a somewhat antiquated term for a slot machine, which tends to "rob" players of their money. The colorful name for our problem comes from a motivating story in which a gambler enters a casino and sits down at a slot machine with multiple levers, or arms, that can be pulled. When pulled, an arm produces a random payout drawn independently of the past. Because the distribution of payouts corresponding to each arm is not listed, the player can learn it only by experimenting. As the gambler learns about the arms' payouts, she faces a dilemma: in the immediate future she expects to earn more by *exploiting* arms that yielded high payouts in the past, but by continuing to *explore* alternative arms she may learn how to earn higher payouts in the future. Can she develop a sequential strategy for pulling arms that balances this tradeoff and maximizes the cumulative payout earned? The following Bernoulli bandit problem is a canonical example.

**Example 1** (*Bernoulli Bandit*) Suppose there are $K$ actions, and when played, any action yields either a success or a failure. Action $k \in \{1, ..., K\}$ produces a success with probability $0 \leq \theta_k \leq 1$. The success probabilities $(\theta_1, .., \theta_K)$ are unknown to the agent, but are fixed over time, and therefore can be learned by experimentation.

The objective, roughly speaking, is to maximize the cumulative number of successes over $T$ periods, where $T$ is relatively large compared to the number of arms $K$.

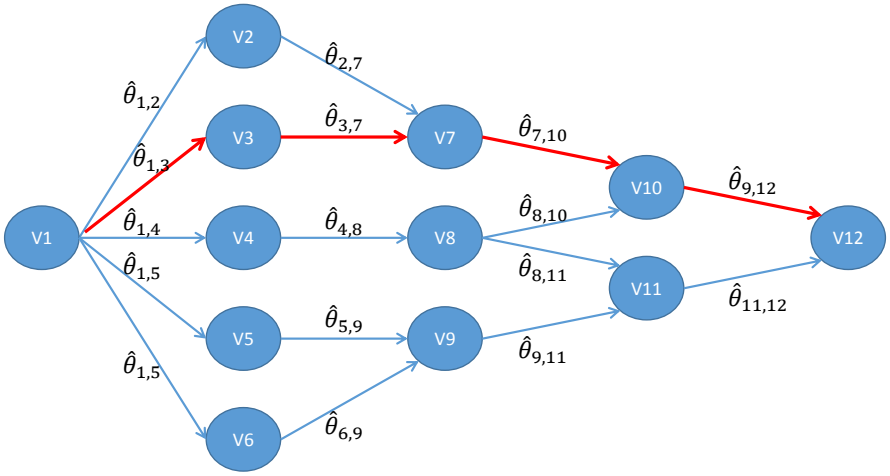
The "arms" in this problem might represent different banner ads that can be displayed on a website. Users arriving at the site are shown versions of the website with different banner ads. A success is associated either with a click on the ad, or with a conversion (a sale of the item being advertised). The parameters $\theta_k$ represent either the click-through-rate or conversion-rate among the population of users who frequent the site. The website hopes to balance exploration and exploitation in order to maximize the total number of successes.

A naive approach to this problem involves allocating some fixed fraction of time periods to exploration and in each such period sampling an arm uniformly at random, while aiming to select successful actions in other time periods. We will observe that such an approach can be quite wasteful even for the simple Bernoulli bandit problem described above and can fail completely for more complicated problems.

Problems like the Bernoulli bandit described above have been studied in the decision sciences since the second world war, as they crystallize the fundamental trade-off between exploration and exploitation in sequential decision making. But the information revolution has created significant new opportunities and challenges, which have spurred a particularly intense interest in this problem in recent years. To understand this, let us contrast the Internet advertising example given above with the problem of choosing a banner ad to display on a highway. A physical banner ad might be changed only once every few months, and once posted will be seen by every individual who drives on the road. There is value to experimentation, but data is limited, and the cost of of trying a potentially ineffective ad is enormous. Online, a different banner ad can be shown to each individual out of a large pool of users, and data from each such interaction is stored. Small-scale experiments are now a core tool at most leading Internet companies.

Our interest in this problem is motivated by this broad phenomenon. Machine learning is increasingly used to make rapid data-driven decisions. While standard algorithms in supervised machine learning learn passively from historical data, these systems often drive the generation of their own training data through interacting with users. An online recommendation system, for example, uses historical data to optimize current recommendations, but the outcomes of these recommendations are then fed back into the system and used to improve future recommendations. As a result, there is enormous potential benefit in the design of algorithms that not only learn from past data, but also explore systemically to generate useful data that improves future performance. There are significant challenges in extending algorithms designed to address Example 1 to treat more realistic and complicated decision problems. To understand some of these challenges, consider the problem of learning by experimentation to solve a shortest path problem.

**Example 2** (Online Shortest Path) An agent commutes from home to work every morning. She would like to commute along the path that requires the least average travel time, but she is uncertain of the travel time along different routes. How can she learn efficiently and minimize the total travel time over a large number of trips?




We can formalize this as a shortest path problem on a graph $G = (V, E)$ with vertices $V = \{1, ..., N\}$ and edges $E$. Here vertex 1 is the source (her home) and vertex $N$ is the destination (work). Each vertex can be thought of as an intersection, and for two vertices $i, j \in V$, an edge $(i, j) \in E$ is present if there is a direct road connecting the two intersections. Suppose that traveling along an edge $e \in E$ requires time $\theta_e$ on average. If these parameters were known, the agent would select a path $(e_1, .., e_n)$, consisting of a sequence of adjacent edges connecting vertices 1 and $N$, such that the expected total time $\theta_{e_1} + ... + \theta_{e_n}$ is minimized. Instead, she chooses paths in a sequence of periods. In period $t$, the realized time $y_{t,e}$ to traverse edge $e$ is drawn independently from a distribution with mean $\theta_e$. The agent sequentially chooses a path $x_t$, observes the realized travel time $(y_{t,e})_{e \in x_t}$ along each edge in the path, and incurs cost $c_t = \sum_{e \in x_t} y_{t,e}$ equal to the total travel time. By exploring intelligently, she hopes to minimize cumulative travel time $\sum_{t=1}^{T} c_t$ over a large number of periods $T$.

This problem is conceptually similar to the Bernoulli bandit in Example 1, but here the number of actions is the number of paths in the graph, which generally scales exponentially in the number of edges. This raises substantial challenges. For moderate sized graphs, trying each possible path would require a prohibitive number of samples, and algorithms that require enumerating and searching through the set of all paths to reach a decision will be computationally intractable. An efficient approach therefore needs to leverage the statistical and computational structure of problem.

In this model, the agent observes the travel time along each edge traversed in a given period. Other feedback models are also natural: the agent might start a timer as she leaves home and checks it once she arrives, effectively only tracking the total travel time of the chosen path. This is closer to the Bernoulli bandit model, where only the realized reward (or cost) of the chosen arm was observed. We have also taken the random edge-delays $y_{t,e}$ to be independent, conditioned on $\theta_e$. A more realistic model might treat these as correlated random variables, reflecting that neighboring roads are likely to be congested at the same time. Rather than design a specialized algorithm for each possible statistical model, we seek a general approach to exploration that accommodates flexible modeling and works for a broad

array of problems. We will see that Thompson sampling accommodates such flexible modeling, and offers an elegant and efficient approach to exploration in a wide range of structured decision problems, including the shortest path problem described here.

Thompson sampling was first proposed in 1933 [1, 2] for allocating experimental effort in two-armed bandit problems arising in clinical trials. The algorithm was large largely ignored in the academic literature until recently, although it was independently rediscovered several times in the interim [3, 4] as an effective heuristic. Now, more than eight decades after it was introduced, Thompson sampling has seen a surge of interest among industry practitioners and academics. This was spurred partly by two influential articles that displayed the algorithm's strong empirical performance [5, 6]. In the subsequent five years, the literature on Thompson sampling has grown rapidly. Adaptations of Thompson sampling have now been successfully applied in a wide variety of domains, including revenue management [7], marketing [8], Monte Carlo tree search [9], A/B testing [10], Internet advertising [10, 11, 12], recommendation systems [13], hyperparameter tuning [14], and arcade games [15]; and have been used at several companies, including Microsoft [10], Google [6, 16], LinkedIn [11, 12], Twitter, Netflix, and Adobe.

The objective of this tutorial is to explain when, why, and how to apply Thompson sampling. A range of examples are used to demonstrate how the algorithm can be used to solve interesting problems and provide clear insight into why it works and when it offers substantial benefit over naive alternatives. The tutorial also provides guidance on approximations to Thompson sampling that can simplify computation as well as practical considerations like prior distribution specification, safety constraints and nonstationarity. Accompanying this tutorial we also release a Python package[1] that reproduces all experiments and figures in this paper [17]. This resource is valuable not only for reproducible research, but also as a reference implementation that may help practioners build intuition for how to practically implement some of the ideas and algorithms we outline in this paper. A concluding section highlights settings where Thompson sampling performs poorly and discusses alternative approaches studied in recent literature. As a baseline and backdrop for our discussion of Thompson sampling, we begin with an alternative approach that does not actively explore.

## 2 Greedy Decisions

*Greedy* algorithms serve as perhaps the simplest and most common approach to online decision problems. The following two steps are taken to generate each action: (1) estimate a model from historical data and (2) select the action that is optimal for the estimated model, breaking ties in an arbitrary manner. Such an algorithm is greedy in the sense that an action is chosen solely to maximize immediate reward. Figure 1 illustrates such a scheme. At each time $t$, a supervised learning algorithm fits a model to historical data pairs $\mathbb{H}_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$, generating an estimate $\hat{\theta}$ of model parameters. The resulting model can then be used to predict the reward $r_t = r(y_t)$ from applying action $x_t$. Here, $y_t$ is an observed outcome, while $r$ is a known function that represents the agent's preferences. Given estimated model parameters $\hat{\theta}$, an optimization algorithm selects the action $x_t$ that maximizes expected reward, assuming that $\theta = \hat{\theta}$. This action is then applied to the exogenous system and an outcome $y_t$ is observed.

A shortcoming of the greedy approach, which can severely curtail performance, is that it does not actively explore. To understand this issue, it is helpful to focus on the Bernoulli bandit setting of Example 1. In that context, the observations are rewards, so $r_t = r(y_t) = y_t$. At each time $t$, a greedy algorithm would generate a estimate $\hat{\theta}_k$ of the mean reward for each $k$th action, and select the action that attains the maximum among these estimates.

[1] Full code and documentation is available at `https://github.com/iosband/ts_tutorial`.

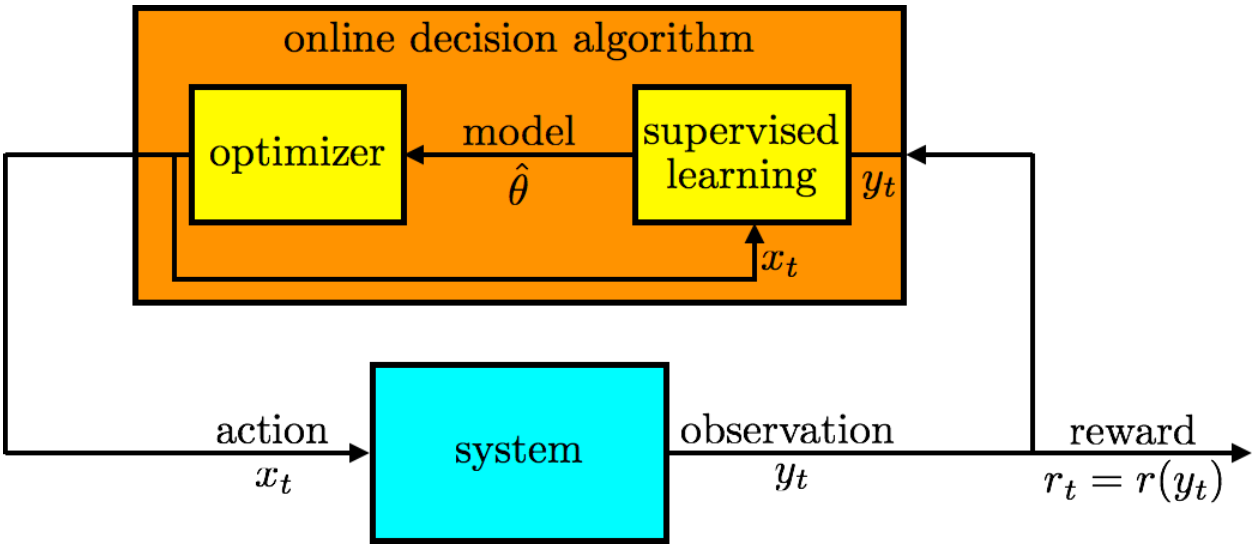



Figure 1: Online decision algorithm.

Suppose there are three actions with mean rewards $\theta \in \mathbb{R}^3$. In particular, each time an action $k$ is selected, a reward of 1 is generated with probability $\theta_k$. Otherwise, a reward of 0 is generated. The mean rewards are not known to the agent. Instead, the agent's beliefs in any given time period about these mean rewards can be expressed in terms of posterior distributions. Suppose that, conditioned on the observed history $\mathbb{H}_{t-1}$, posterior distributions are represented by the probability density functions plotted in Figure 2. These distributions represent beliefs after the agent tries actions 1 and 2 one thousand times each, action 3 five times, receives cumulative rewards of 600, 400, and 2, respectively, and synthesizes these observations with uniform prior distributions over mean rewards of each action. They indicate that the agent is confident that mean rewards for actions 1 and 2 are close to their expectations of 0.6 and 0.4. On the other hand, the agent is highly uncertain about the mean reward of action 3, though he expects 0.4.

The greedy algorithm would select action 1, since that offers the maximal expected mean reward. Since the uncertainty around this expected mean reward is small, observations are unlikely to change the expectation substantially, and therefore, action 1 is likely to be selected *ad infinitum*. It seems reasonable to avoid action 2, since it is extremely unlikely that $\theta_2 > \theta_1$. On the other hand, if the agent plans to operate over many time periods, it should try action 3. This is because there is some chance that $\theta_3 > \theta_1$, and if this turns out to be the case, the agent will benefit from learning that and applying action 3. To learn whether $\theta_3 > \theta_1$, the agent needs to try action 3, but the greedy algorithm will unlikely ever do that. The algorithm fails to account for uncertainty in the mean reward of action 3, which should entice the agent to explore and learn about that action.

*Dithering* is a common approach to exploration that operates through randomly perturbing actions that would be selected by a greedy algorithm. One version of dithering, called *$\epsilon$-greedy exploration*, applies the greedy action with probability $1 - \epsilon$ and otherwise selects an action uniformly at random. Though this form of exploration can improve behavior relative to a purely greedy approach, it wastes resources by failing to "write off" actions regardless of how unlikely they are to be optimal. To understand why, consider again the posterior distributions of Figure 2. Action 2 has almost no chance of being optimal, and therefore, does not deserve experimental trials, while the uncertainty surrounding action 3 warrants exploration. However,

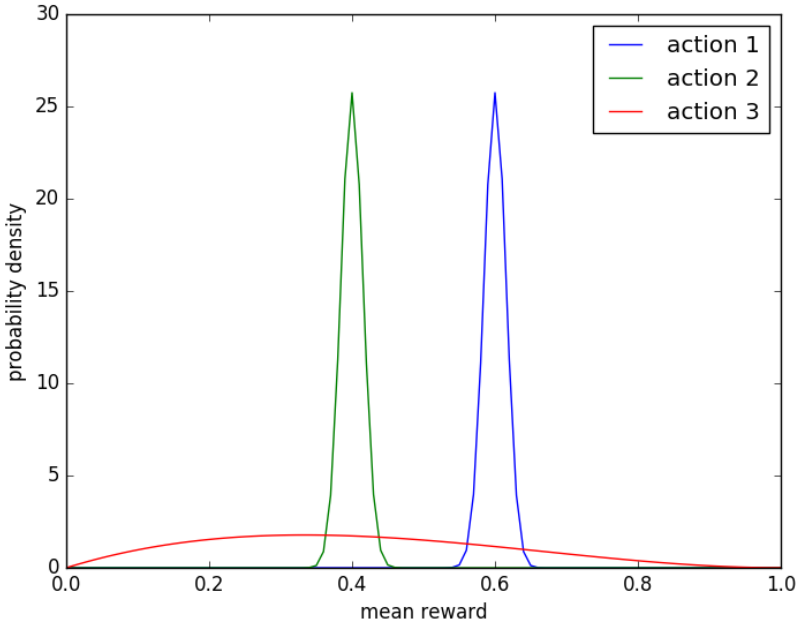



Figure 2: Probability density functions over mean rewards.

$\epsilon$-greedy exploration would allocate an equal number of experimental trials to each action. Though only half of the exploratory actions are wasted in this example, the issue is exacerbated as the number of possible actions increases. Thompson sampling, introduced more than eight decades ago [1], provides an alternative to dithering that more intelligently allocates exploration effort.

# 3 Thompson Sampling for the Bernoulli Bandit

To digest how Thompson sampling works, it is helpful to begin with a simple context that builds on the Bernoulli bandit of Example 1 and incorporates a Bayesian model to represent uncertainty.

**Example 3** (Beta-Bernoulli Bandit) Recall the Bernoulli bandit of Example 1. There are $K$ actions. When played, an action $k$ produces a reward of one with probability $\theta_k$ and a reward of zero with probability $1-\theta_k$. Each $\theta_k$ can be interpreted as an action's success probability or mean reward. The mean rewards $\theta = (\theta_1, ..., \theta_K)$ are unknown, but fixed over time. In the first period, an action $x_1$ is applied, and a reward $r_1 \in \{0, 1\}$ is generated with success probability $\mathbb{P}(r_1 = 1|x_1, \theta) = \theta_{x_1}$. After observing $r_1$, the agent applies another action $x_2$, observes a reward $r_2$, and this process continues.

Let the agent begin with an independent prior belief over each $\theta_k$. Take these priors to be beta-distributed with parameters $\alpha = (\alpha_1, \ldots, \alpha_K)$ and $\beta \in (\beta_1, \ldots, \beta_K)$. In particular, for each action $k$, the prior probability density function of $\theta_k$ is

$$p(\theta_k) = \frac{\Gamma(\alpha_k + \beta_k)}{\Gamma(\alpha_k)\Gamma(\beta_k)} \theta_k^{\alpha_k - 1}(1 - \theta_k)^{\beta_k - 1},$$

where $\Gamma$ denotes the gamma function. As observations are gathered, the distribution is updated according to Bayes' rule. It is particularly convenient to work with beta distributions because of their conjugacy properties. In particular, each action's posterior distribution is also beta with parameters that can be updated according to a simple rule:

$$(\alpha_k, \beta_k) \leftarrow \begin{cases} (\alpha_k, \beta_k) & \text{if } x_t \neq k \\ (\alpha_k, \beta_k) + (r_t, 1 - r_t) & \text{if } x_t = k. \end{cases}$$

Note that for the special case of $\alpha_k = \beta_k = 1$, the prior $p(\theta_k)$ is uniform over $[0, 1]$. Note that only the parameters of a selected arm are updated. The parameters $(\alpha_k, \beta_k)$ are sometimes called pseudo-counts, since $\alpha_k$ or $\beta_k$ increases by one with each observed success or failure, respectively. A beta distribution with parameters $(\alpha_k, \beta_k)$ has mean $\alpha_k/(\alpha_k + \beta_k)$, and the distribution becomes more concentrated as $\alpha_k + \beta_k$ grows. Figure 2 plots probability density functions of beta distributions with parameters $(\alpha_1, \beta_1) = (600, 400)$, $(\alpha_2, \beta_2) = (400, 600)$, and $(\alpha_3, \beta_3) = (4, 6)$.

Algorithm 2 presents a greedy algorithm for the beta-Bernoulli bandit. In each time period $t$, the algorithm generates an estimate $\hat{\theta}_k = \alpha_k/(\alpha_k + \beta_k)$, equal to its current expectation of the success probability $\theta_k$. The action $x_t$ with the largest estimate $\hat{\theta}_k$ is then applied, after which a reward $r_t$ is observed and the distribution parameters $\alpha_{x_t}$ and $\beta_{x_t}$ are updated.

**Algorithm 1** BernGreedy$(K, \alpha, \beta)$

**for** $t = 1, 2, \ldots$ **do**
  #estimate model:
  **for** $k = 1, \ldots, K$ **do**
    $\hat{\theta}_k \leftarrow \alpha_k/(\alpha_k + \beta_k)$
  **end for**

  #select and apply action:
  $x_t \leftarrow \text{argmax}_k\, \hat{\theta}_k$
  Apply $x_t$ and observe $r_t$

  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$
**end for**

**Algorithm 2** BernThompson$(K, \alpha, \beta)$

**for** $t = 1, 2, \ldots$ **do**
  #sample model:
  **for** $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_k \sim \text{beta}(\alpha_k, \beta_k)$
  **end for**

  #select and apply action:
  $x_t \leftarrow \text{argmax}_k\, \hat{\theta}_k$
  Apply $x_t$ and observe $r_t$

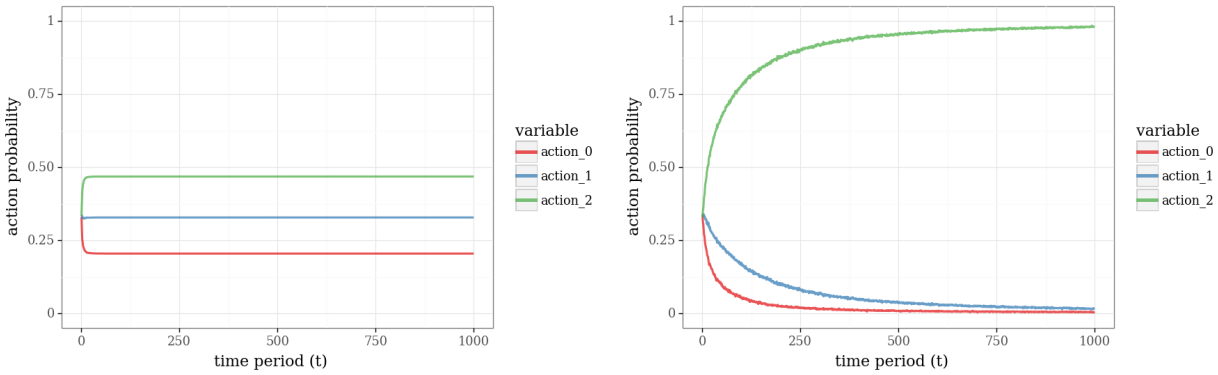
  #update distribution:
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$
**end for**

Thompson sampling, specialized to the case of a beta-Bernoulli bandit, proceeds similarly, as presented in Algorithm 2. The only difference is that the success probability estimate $\hat{\theta}_k$ is randomly sampled from the posterior distribution, which is a beta distribution with parameters $\alpha_k$ and $\beta_k$, rather than taken to be the expectation $\alpha_k/(\alpha_k + \beta_k)$. To avoid a common misconception, it is worth emphasizing Thompson sampling does **not** sample $\hat{\theta}_k$ from the posterior distribution of the binary value $y_t$ that would be observed if action $k$ is selected. In particular, $\hat{\theta}_k$ represents a statistically plausible success probability rather than a statistically plausible observation.

To understand how Thompson sampling improves on greedy actions with or without dithering, recall the three armed Bernoulli bandit with posterior distributions illustrated in Figure 2. In this context, a greedy action would forgo the potentially valuable opportunity to learn

about action 3. With dithering, equal chances would be assigned to probing actions 2 and 3, though probing action 2 is virtually futile since it is extremely unlikely to be optimal. Thompson sampling, on the other hand would sample actions 1, 2, or 3, with probabilities approximately equal to 0.82, 0, and 0.18, respectively. In each case, this is the probability that the random estimate drawn for the action exceeds those drawn for other actions. Since these estimates are drawn from posterior distributions, each of these probabilities is also equal to the probability that the corresponding action is optimal, conditioned on observed history. As such, Thompson sampling explores to resolve uncertainty where there is a chance that resolution will help the agent identify the optimal action, but avoids probing where feedback would not be helpful.

It is illuminating to compare simulated behavior of Thompson sampling to that of a greedy algorithm. Consider a three-armed beta-Bernoulli bandit with mean rewards $\theta_1 = 0.9$, $\theta_2 = 0.8$, and $\theta_3 = 0.7$. Let the prior distribution over each mean reward be uniform. Figure 3 plots results based on ten thousand independent simulations of each algorithm. Each simulation is over one thousand time periods. In each simulation, actions are randomly rank-ordered for the purpose of tie-breaking so that the greedy algorithm is not biased toward selecting any particular action. Each data point represents the fraction of simulations for which a particular action is selected at a particular time.




(a) greedy algorithm

(b) Thompson sampling

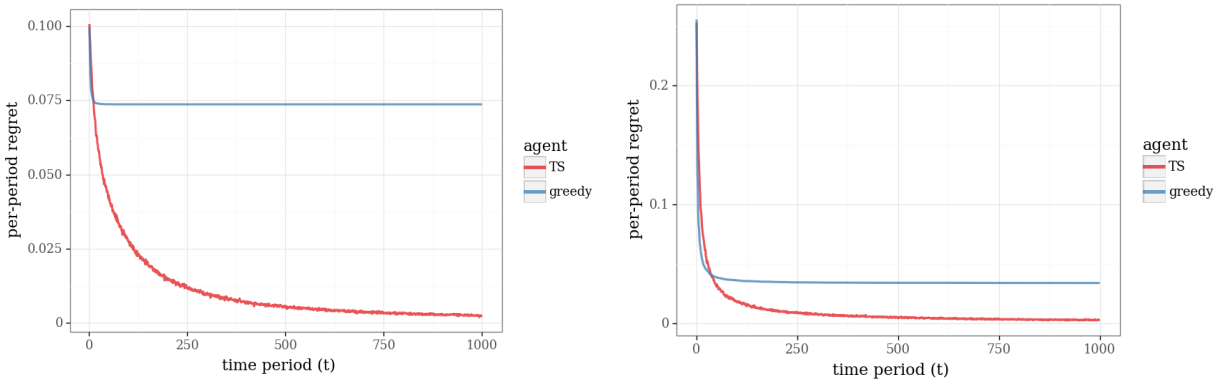
Figure 3: Probability that the greedy algorithm and Thompson sampling selects an action.

From the plots, we see that the greedy algorithm does not always converge on action 1, which is the optimal action. This is because the algorithm can get stuck, repeatedly applying a poor action. For example, suppose the algorithm applies action 3 over the first couple time periods and receives a reward of 1 on both occasions. The algorithm would then continue to select action 3, since the expected mean reward of either alternative remains at 0.5. With repeated selection of action 3, the expected mean reward converges to the true value of 0.7, which reinforces the agent's commitment to action 3. Thompson sampling, on the other hand, learns to select action 1 within the thousand periods. This is evident from the fact that, in an overwhelmingly large fraction of simulations, Thompson sampling selects action 1 in the final period.

The performance of online decision algorithms is often studied and compared through plots of regret. The *per-period regret* of an algorithm over a time period $t$ is the difference between the mean reward of an optimal action and the action selected by the algorithm. For the Bernoulli bandit problem, we can write this as $\text{regret}_t(\theta) = \max_k \theta_k - \theta_{x_t}$. Figure 4a plots per-period regret realized by the greedy algorithm and Thompson sampling, again averaged over ten thousand simulations. The average per-period regret of Thompson sampling vanishes as time progresses. That is not the case for the greedy algorithm.

Comparing algorithms with fixed mean rewards raises questions about the extent to which

the results depend on the particular choice of $\theta$. As such, it is often useful to also examine regret averaged over plausible values of $\theta$. A natural approach to this involves sampling many instances of $\theta$ from the prior distributions and generating an independent simulation for each. Figure 4b plots averages over ten thousand such simulations, with each action reward sampled independently from a uniform prior for each simulation. Qualitative features of these plots are similar to those we inferred from Figure 4a, though regret in Figure 4a is generally smaller over early time periods and larger over later time periods, relative to Figure 4b. The smaller regret in early time periods is due to the fact that with $\theta = (0.9, 0.8, 0.7)$, mean rewards are closer than for a typical randomly sampled $\theta$, and therefore the regret of randomly selected actions is smaller. The reduction in later time periods is also a consequence of proximity among rewards with $\theta = (0.9, 0.8, 0.7)$. In this case, the difference is due to the fact that it takes longer to differentiate actions than it would for a typical randomly sampled $\theta$.




(a) $\theta = (0.9, 0.8, 0.7)$

(b) average over random $\theta$

Figure 4: Regret from applying greedy and Thompson sampling algorithms to the three-armed Bernoulli bandit.

# 4 General Thompson Sampling

Thompson sampling can be applied fruitfully to a broad array of online decision problems beyond the Bernoulli bandit, and we now consider a more general setting. Suppose the agent applies a sequence of actions $x_1, x_2, x_3, \ldots$ to a system, selecting each from a set $\mathcal{X}$. This action set could be finite, as in the case of the Bernoulli bandit, or infinite. After applying action $x_t$, the agent observes an outcome $y_t$, which the system randomly generates according to a conditional probability measure $q_\theta(\cdot|x_t)$. The agent enjoys a reward $r_t = r(y_t)$, where $r$ is a known function. The agent is initially uncertain about the value of $\theta$ and represents his uncertainty using a prior distribution $p$.

Algorithms 3 and 4 present greedy and Thompson sampling approaches in an abstract form that accommodates this very general problem. The two differ in the way they generate model parameters $\hat{\theta}$. The greedy algorithm takes $\hat{\theta}$ to be the expectation of $\theta$ with respect to the distribution $p$, while Thompson sampling draws a random sample from $p$. Both algorithms then apply actions that maximize expected reward for their respective models. Note that, if there are a finite set of possible observations $y_t$, this expectation is given by

$$\mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x] = \sum_o q_{\hat{\theta}}(o|x) r(o). \tag{4.1}$$

The distribution $p$ is updated by conditioning on the realized observation $\hat{y}_t$. If $\theta$ is restricted to values from a finite set, this conditional distribution can be written by Bayes rule as

$$\mathbb{P}_{p,q}(\theta = u | x_t, y_t) = \frac{p(u) q_u(y_t|x_t)}{\sum_v p(v) q_v(y_t|x_t)}. \tag{4.2}$$

**Algorithm 3** Greedy$(\mathcal{X}, p, q, r)$

**for** $t = 1, 2, \ldots$ **do**
  #estimate model:
  $\hat{\theta} \leftarrow \mathbb{E}_p[\theta]$

  #select and apply action:
  $x_t \leftarrow \text{argmax}_{x \in \mathcal{X}} \mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x]$
  Apply $x_t$ and observe $y_t$

  #update distribution:
  $p \leftarrow \mathbb{P}_{p,q}(\theta \in \cdot | x_t, y_t)$
**end for**

**Algorithm 4** Thompson$(\mathcal{X}, p, q, r)$

**for** $t = 1, 2, \ldots$ **do**
  #sample model:
  Sample $\hat{\theta} \sim p$

  #select and apply action:
  $x_t \leftarrow \text{argmax}_{x \in \mathcal{X}} \mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x]$
  Apply $x_t$ and observe $y_t$

  #update distribution:
  $p \leftarrow \mathbb{P}_{p,q}(\theta \in \cdot | x_t, y_t)$
**end for**

The Bernoulli bandit with a beta prior serves as a special case of this more general formulation. In this special case, the set of actions is $\mathcal{X} = \{1, \ldots, K\}$ and only rewards are observed, so $y_t = r_t$. Observations and rewards are modeled by conditional probabilities $q_\theta(1|k) = \theta_k$ and $q_\theta(0|k) = 1 - \theta_k$. The prior distribution is encoded by vectors $\alpha$ and $\beta$, with probability density function given by:

$$p(\theta) = \prod_{k=1}^{K} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha_k)\Gamma(\beta_k)} \theta_k^{\alpha_k - 1} (1 - \theta_k)^{\beta_k - 1},$$

where $\Gamma$ denotes the gamma function. In other words, under the prior distribution, components of $\theta$ are independent and beta-distributed, with parameters $\alpha$ and $\beta$.

For this problem, the greedy algorithm (Algorithm 3) and Thompson sampling (Algorithm 4) begin each $t$th iteration with posterior parameters $(\alpha_e, \beta_e)$ for $e \in E$. The greedy algorithm sets $\hat{\theta}_e$ to the expected value $\mathbb{E}_p[\theta_e] = \alpha_e/(\alpha_e + \beta_e)$, whereas Thompson sampling randomly draws $\hat{\theta}_e$ from a beta distribution with parameters $(\alpha_e, \beta_e)$. Each algorithm then selects the action $x$ that maximizes $\mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x] = \hat{\theta}_x$. After applying the selected action, a reward $r_t = y_t$ is observed, and belief distribution parameters are updated according to

$$(\alpha, \beta) \leftarrow (\alpha + r_t \mathbf{1}_{x_t}, \beta + (1 - r_t)\mathbf{1}_{x_t}),$$

where $\mathbf{1}_{x_t}$ is a vector with component $x_t$ equal to 1 and all other components equal to 0.

Algorithms 3 and 4 can also be applied to much more complex problems. As an example, let us consider a version of the shortest path problem presented in Example 2.

> **Example 4** (Independent Travel Times) Recall the shortest path problem of Example 2. The model is defined with respect to a directed graph $G = (V, E)$, with vertices $V = \{1, \ldots, N\}$, edges $E$, and mean travel times $\theta \in \mathbb{R}^N$. Vertex 1 is the source and vertex $N$ is the destination. An action is a sequence of distinct edges leading from source to destination. After applying action $x_t$, for each traversed edge $e \in x_t$, the agent observes a travel time $y_{t,e}$ that is independently sampled from a

distribution with mean $\theta_e$. Further, the agent incurs a cost of $\sum_{e \in x_t} y_{t,e}$, which can be thought of as a reward $r_t = -\sum_{e \in x_t} y_{t,e}$.

Consider a prior for which each $\theta_e$ is independent and lognormally-distributed with parameters $\mu_e$ and $\sigma_e^2$. That is, $\ln(\theta_e) \sim N(\mu_e, \sigma_e^2)$ is normally distributed. Hence, $\mathbb{E}[\theta_e] = e^{\mu_e + \sigma_e^2/2}$. Further, take $y_{t,e}|\theta$ to be independent across edges $e \in E$ and lognormally distributed with parameters $\ln \theta_e - \tilde{\sigma}^2/2$ and $\tilde{\sigma}^2$, so that $\mathbb{E}[y_{t,e}|\theta_e] = \theta_e$. Conjugacy properties accommodate a simple rule for updating the distribution of $\theta_e$ upon observation of $y_{t,e}$:

$$(\mu_e, \sigma_e^2) \leftarrow \left( \frac{\frac{1}{\sigma_e^2}\mu_e + \frac{1}{\tilde{\sigma}^2}\left(\ln y_{t,e} + \frac{\tilde{\sigma}^2}{2}\right)}{\frac{1}{\sigma_e^2} + \frac{1}{\tilde{\sigma}^2}}, \frac{1}{\frac{1}{\sigma_e^2} + \frac{1}{\tilde{\sigma}^2}} \right). \tag{4.3}$$

To motivate this formulation, consider an agent who commutes from home to work every morning. Suppose possible paths are represented by a graph $G = (V, E)$. Suppose the agent knows the travel distance $d_e$ associated with each edge $e \in E$ but is uncertain about average travel times. It would be natural for her to construct a prior for which expectations are equal to travel distances. With the lognormal prior, this can be accomplished by setting $\mu_e = \ln d_e - \sigma_e^2/2$. Note that the parameters $\mu_e$ and $\sigma_e^2$ also express a degree of uncertainty; in particular, the prior variance of mean travel time along an edge is $(e^{\sigma_e^2} - 1)d_e^2$.

The greedy algorithm (Algorithm 3) and Thompson sampling (Algorithm 4) can be applied to Example 4 in a computationally efficient manner. Each algorithm begins each $t$th iteration with posterior parameters $(\mu_e, \sigma_e)$ for each $e \in E$. The greedy algorithm sets $\hat{\theta}_e$ to the expected value $\mathbb{E}_p[\theta_e] = e^{\mu_e + \sigma_e^2/2}$, whereas Thompson sampling randomly draws $\hat{\theta}_e$ from a lognormal distribution with parameters $\mu_e$ and $\sigma_e^2$. Each algorithm then selects its action $x$ to maximize $\mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x] = -\sum_{e \in x_t} \hat{\theta}_e$. This can be cast as a deterministic shortest path problem, which can be solved efficiently, for example, via Dijkstra's algorithm. After applying the selected action, an outcome $y_t$ is observed, and belief distribution parameters $(\mu_e, \sigma_e^2)$, for each $e \in E$, are updated according to (4.3).

Figure 6 presents results from applying greedy and Thompson sampling algorithms to Example 4, with the graph taking the form of a binomial bridge, as shown in Figure 5, except with twenty rather than six stages, so there are 184,756 paths from source to destination. Prior parameters are set to $\mu_e = -\frac{1}{2}$ and $\sigma_e^2 = 1$ so that $\mathbb{E}[\theta_e] = 1$, for each $e \in E$, and the conditional distribution parameter is $\tilde{\sigma}^2 = 1$. Each data point represents an average over ten thousand independent simulations.

The plots of regret demonstrate that the performance of Thompson sampling converges quickly to that of the optimal policy, while that is far from true for the greedy algorithm. We also plot results generated by $\epsilon$-greedy exploration, varying $\epsilon$. For each trip, with probability $1-\epsilon$, this algorithm traverses a path produced by a greedy algorithm. Otherwise, the algorithm samples a path randomly. Though this form of exploration can be helpful, the plots demonstrate that learning progresses at a far slower pace than with Thompson sampling. This is because $\epsilon$-greedy exploration is not judicious in how it selects paths to explore. Thompson sampling, on the other hand, orients exploration effort towards informative rather than entirely random paths.

Plots of cumulative travel time relative to optimal offer a sense for the fraction of driving time wasted due to lack of information. Each point plots an average of the ratio between the time incurred over some number of days and the minimal expected travel time given $\theta$. With Thompson sampling, this converges to one at a respectable rate. The same can not be said for $\epsilon$-greedy approaches.

Algorithm 4 can be applied to problems with complex information structures, and there is often substantial value to careful modeling of such structures. As an example, we consider a

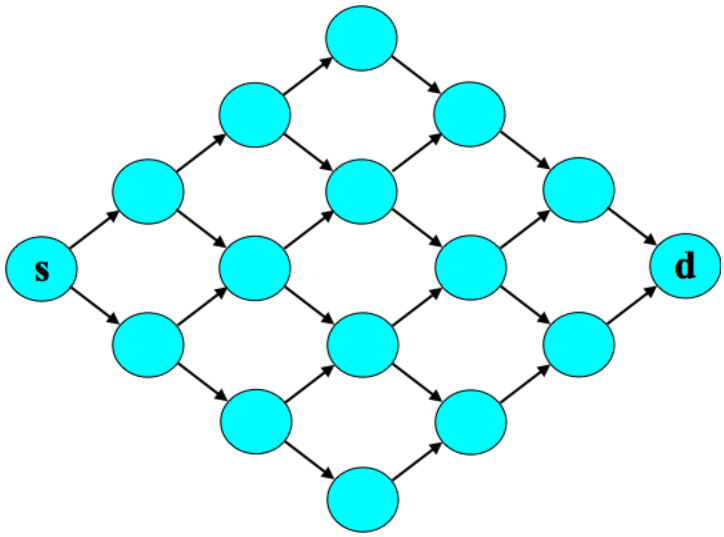
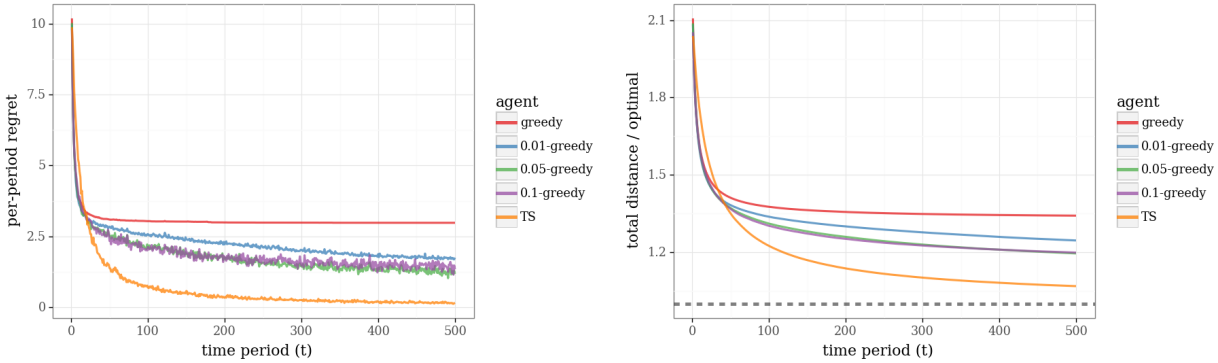



Figure 5: A binomial bridge with six stages.




(a) regret

(b) cumulative travel time relative to optimal

Figure 6: Performance of Thompson sampling and $\epsilon$-greedy algorithms in the shortest path problem.

more complex variation of the binomial bridge example.

**Example 5** (Correlated Travel Times) As with Example 4, let each $\theta_e$ be independent and lognormally-distributed with parameters $\mu_e$ and $\sigma_e^2$. Let the observation distribution be characterized by

$$y_{t,e} = \zeta_{t,e}\eta_t\nu_{t,\ell(e)}\theta_e,$$

where each $\zeta_{t,e}$ represents an idiosyncratic factor associated with edge $e$, $\eta_t$ represents a factor that is common to all edges, $\ell(e)$ indicates whether edge $e$ resides in the lower half of the binomial bridge, and $\nu_{t,0}$ and $\nu_{t,1}$ represent factors that bear a common influence on edges in the upper and lower halves, respectively. We take each $\zeta_{t,e}$, $\eta_t$, $\nu_{t,0}$, and $\nu_{t,1}$ to be independent lognormally distributed with parameters $-\tilde{\sigma}^2/6$ and $\tilde{\sigma}^2/3$. The distributions of the shocks $\zeta_{t,e}$, $\eta_t$, $\nu_{t,0}$ and $\nu_{t,1}$ are known, and only the parameters $\theta_e$ corresponding to each individual edge must be learned through experimentation. Note that, given these parameters, the marginal distribution of $y_{t,e}|\theta$ is identical to that of Example 4, though the joint distribution over $y_t|\theta$ differs.

The common factors induce correlations among travel times in the binomial bridge: $\eta_t$ models the impact of random events that influence traffic conditions everywhere, like the day's weather, while $\nu_{t,0}$ and $\nu_{t,1}$ each reflect events that bear influence only on traffic conditions along edges in half of the binomial bridge. Though mean edge travel times are independent under the prior, correlated observations induce dependencies in posterior distributions.

Conjugacy properties again facilitate efficient updating of posterior parameters. Let $\phi, z_t \in \mathbb{R}^N$ be defined by

$$\phi_e = \ln \theta_e \qquad \text{and} \qquad z_{t,e} = \begin{cases} \ln y_{t,e} & \text{if } e \in x_t \\ 0 & \text{otherwise.} \end{cases}$$

Note that it is with some abuse of notation that we index vectors and matrices using edge indices. Define a $|x_t| \times |x_t|$ covariance matrix $\tilde{\Sigma}$ with elements

$$\tilde{\Sigma}_{e,e'} = \begin{cases} \tilde{\sigma}^2 & \text{for } e = e' \\ 2\tilde{\sigma}^2/3 & \text{for } e \neq e', \ell(e) = \ell(e') \\ \tilde{\sigma}^2/3 & \text{otherwise,} \end{cases}$$

for $e, e' \in x_t$, and a $N \times N$ concentration matrix

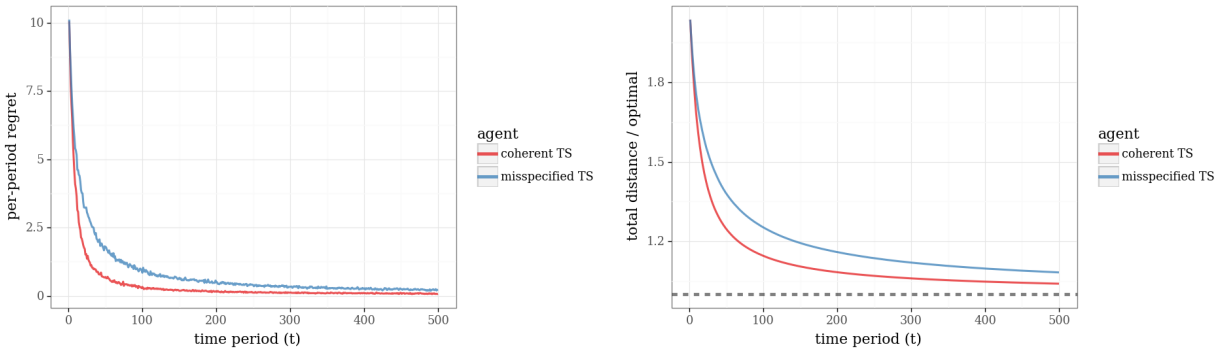
$$\tilde{C}_{e,e'} = \begin{cases} \tilde{\Sigma}^{-1}_{e,e'} & \text{if } e, e' \in x_t \\ 0 & \text{otherwise,} \end{cases}$$

for $e, e' \in E$. Then, the posterior distribution of $\phi$ is normal with a mean vector $\mu$ and covariance matrix $\Sigma$ that can be updated according to

$$(\mu, \Sigma) \leftarrow \left( \left( \Sigma^{-1} + \tilde{C} \right)^{-1} \left( \Sigma^{-1}\mu + \tilde{C} z_t \right), \left( \Sigma^{-1} + \tilde{C} \right)^{-1} \right). \tag{4.4}$$

Thompson sampling (Algorithm 4) can again be applied in a computationally efficient manner. Each $t$th iteration begins with posterior parameters $\mu \in \mathbb{R}^N$ and $\Sigma \in \mathbb{R}^{N \times N}$. The sample $\hat{\theta}$ can be drawn by first sampling a vector $\hat{\phi}$ from a normal distribution with mean $\mu$ and covariance matrix $\Sigma$, and then setting $\hat{\theta}_e = \hat{\phi}_e$ for each $e \in E$. An action $x$ is selected to maximize $\mathbb{E}_{q_{\hat{\theta}}}[r(y_t)|x_t = x] = -\sum_{e \in x_t} \hat{\theta}_e$, using Djikstra's algorithm or an alternative. After applying the selected action, an outcome $y_t$ is observed, and belief distribution parameters $(\mu, \Sigma)$ are updated according to (4.4).

Figure 7 plots results from applying Thompson sampling to Example 5, again with the binomial bridge, $\mu_e = -\frac{1}{2}$, $\sigma_e^2 = 1$, and $\tilde{\sigma}^2 = 1$. Each data point represents an average over ten thousand independent simulations. Despite model differences, an agent can pretend that observations made in this new context are generated by the model described in Example 4. In particular, the agent could maintain an independent lognormal posterior for each $\theta_e$, updating parameters $(\mu_e, \sigma_e^2)$ as though each $y_{t,e}|\theta$ is independently drawn from a lognormal distribution. As a baseline for comparison, Figure 7 additionally plots results from application of this approach, which we will refer to here as *misspecified Thompson sampling*. The comparison demonstrates substantial improvement that results from accounting for interdependencies among edge travel times, as is done by what we refer to here as *coherent Thompson sampling*. Note that we have assumed here that the agent must select a path before initiating each trip. In particular, while the agent may be able to reduce travel times in contexts with correlated delays by adjusting the path during the trip based on delays experienced so far, our model does not allow this behavior.




(a) regret

(b) cumulative travel time relative to optimal

Figure 7: Performance of two versions of Thompson sampling in the shortest path problem with correlated travel times.

# 5 Approximations

Conjugacy properties in the Bernoulli bandit and shortest path examples that we have considered so far facilitated simple and computationally efficient Bayesian inference. Indeed, computational efficiency can be an important consideration when formulating a model. However, many practical contexts call for more complex models for which exact Bayesian inference is computationally intractable. Fortunately, there are reasonably efficient and accurate methods that can be used to approximately sample from posterior distributions.

In this section we discuss four approaches to approximate posterior sampling: Gibbs sampling, Langevin Monte Carlo, sampling from a Laplace approximation, and the bootstrap. Such methods are called for when dealing with problems that are not amenable to efficient Bayesian inference. As an example, we consider a variation of the online shortest path problem.

> **Example 6** (Binary Feedback) Consider Example 5, except with deterministic travel times and noisy binary observations. Let the graph represent a binomial bridge with $M$ stages. Let each $\theta_e$ be independent and gamma-distributed with $\mathbb{E}[\theta_e] = 1$, $\mathbb{E}[\theta_e^2] = 1.5$, and observations be generated according to
>
> $$y_t|\theta \sim \begin{cases} 1 & \text{with probability } \frac{1}{1+\exp\left(\sum_{e\in x_t}\theta_e - M\right)} \\ 0 & \text{otherwise.} \end{cases}$$
>
> We take the reward to be the rating $r_t = y_t$. This information structure could be used to model, for example, an Internet route recommendation service. Each day, the system recommends a route $x_t$ and receives feedback $y_t$ from the driver, expressing whether the route was desirable. When the realized travel time $\sum_{e\in x_t}\theta_e$ falls short of the prior expectation $M$, the feedback tends to be positive, and vice versa.

This new model does not enjoy conjugacy properties leveraged in Section 4 and is not amenable to efficient exact Bayesian inference. However, the problem may be addressed via approximation methods. To illustrate, Figure 8 plots results from application of two approximate versions of Thompson sampling to an online shortest path problem on a twenty-stage binomial bridge with binary feedback. The algorithms leverage the Laplace approximation and the bootstrap, two approaches we will discuss, and the results demonstrate effective learning, in the sense that regret vanishes over time.

In the remainder of this section, we will describe several approaches to approximate Thompson sampling. It is worth mentioning that we do not cover an exhaustive list, and further, our

descriptions do not serve as comprehensive or definitive treatments of each approach. Rather, our intent is to offer simple descriptions that convey key ideas that may be extended or combined to serve needs arising in any specific application.

Throughout this section, let $f_{t-1}$ denote the posterior density of $\theta$ conditioned on the history of observations $\mathbb{H}_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$. Thompson sampling generates an action $x_t$ by sampling a parameter vector $\hat{\theta}$ from $f_{t-1}$ and solving for the optimal path under $\hat{\theta}$. The methods we describe generate a sample $\hat{\theta}$ whose distribution approximates the posterior $\hat{f}_{t-1}$, which enables approximate implementations of Thompson sampling when exact posterior sampling is infeasible.

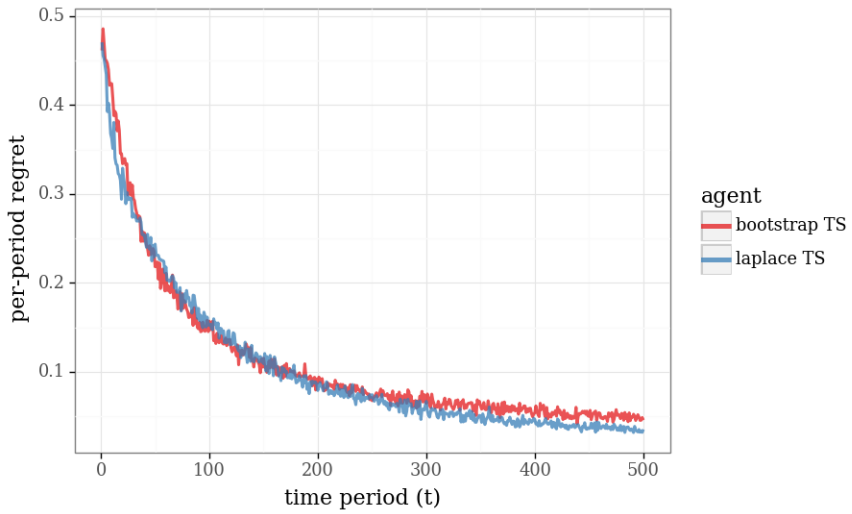



Figure 8: Regret experienced by approximation methods applied to the path recommendation problem with binary feedback.

## 5.1 Gibbs Sampling

Gibbs sampling is a general Markov chain Monte Carlo (MCMC) algorithm for drawing approximate samples from multivariate probability distributions. It produces a sequence of sampled parameters $(\hat{\theta}^n : n = 0, 2 \ldots)$ forming a Markov chain with stationary distribution $f_{t-1}$. Under reasonable technical conditions, the limiting distribution of this Markov chain is its stationary distribution, and the distribution of $\hat{\theta}^n$ converges to $f_{t-1}$.

Gibbs sampling starts with an initial guess $\hat{\theta}^0$. Iterating over sweeps $n = 1, \ldots, N$, for each $n$th sweep, the algorithm iterates over the components $k = 1, \ldots, K$, for each $k$ generating a one-dimensional marginal distribution

$$f_{t-1}^{n,k}(\theta_k) \propto f_{t-1}((\hat{\theta}_1^n, \ldots, \hat{\theta}_{k-1}^n, \theta_k, \hat{\theta}_{k+1}^{n-1}, \ldots, \hat{\theta}_K^{n-1})),$$

and sampling the $k$th component according to $\hat{\theta}_k^n \sim f_{t-1}^{n,k}$. After $N$ of sweeps, the prevailing vector $\hat{\theta}^N$ is taken to be the approximate posterior sample. We refer to [18] for a more thorough introduction to the algorithm.

Gibbs sampling applies to a broad range of problems, and is often computationally viable even when sampling from $f_{t-1}$ is not. This is because sampling from a one-dimensional distribution is simpler. That said, for complex problems, Gibbs sampling can still be computationally demanding. This is the case, for example, with our path recommendation problem with binary

feedback. In this context, it is easy to implement a version of Gibbs sampling that generates a close approximation to a posterior samples within well under a minute. However, running thousands of simulations each over hundreds of time periods can be quite time-consuming. As such, we turn to more efficient approximation methods.

## 5.2 Langevin Monte Carlo

We now describe an alternative Markov chain Monte Carlo method that uses gradient information about the target distribution. Let $g(\phi)$ denote a log-concave probability density function over $\mathbb{R}^K$ from which we wish to sample. Suppose that $\ln g(\phi)$ is differentiable and its gradients are efficiently computable. Arising first in physics, Langevin daynmics refer to the diffusion process

$$d\phi_t = \nabla \ln g(\phi_t)dt + \sqrt{2}dB_t \tag{5.1}$$

where $B_t$ is a standard Brownian motion process. This process has $g(\phi)$ as its unique stationary distribution, and under reasonable technical conditions, the distribution of $\phi_t$ converges rapidly to this stationary distribution [19]. Therefore simulating the process (5.1) provides a means of approximately sampling from $g(\phi)$.

Typically, one instead implements a Euler discretization of this stochastic differential equation

$$\phi_{n+1} = \phi_n + \epsilon \nabla \ln g(\phi_n) + \sqrt{2\epsilon} W_n \qquad n \in \mathbb{N}, \tag{5.2}$$

where $W_1, W_2, \cdots$ are i.i.d. standard normal random variables and $\epsilon > 0$ is a small step size. Like a gradient ascent method, under this method $\phi_n$ tends to drift in directions of increasing density $g(\phi_n)$. However, random Gaussian noise $W_n$ is injected at each step so that, for large $n$, the position of $\phi_n$ is random and captures the uncertainty in the distribution $g$. A number of papers establish rigorous guarantees for the rate at which this Markov chain converges to its stationary distribution [20, 21, 22, 23]. These papers typically require $\epsilon$ is sufficiently small, or that a decaying sequence of step-sizes $(\epsilon_1, \epsilon_2, \cdots)$ is used. Recent work [24, 25] has studied *stochastic gradient* Langevin Monte Carlo, which uses sampled minibatches of data to compute approximate rather than exact gradients.

For the path recommendation problem in Example 6, we have found that the log posterior density becomes ill conditioned in later time periods. For this reason, gradient ascent converges very slowly to the posterior mode. Effective optimization methods must somehow leverage second order information. Similarly, due to poor conditioning, we may need to choose an extremely small step-size $\epsilon$, causing the Markov chain in 5.2 to mix slowly. We have found that preconditioning substantially improves performance. Langevin MCMC can be implemented with a symmetric positive definite preconditioning matrix $M$ by simulating the Markov chain

$$\phi_{n+1} = \phi_n + \epsilon M \nabla \ln g(\phi_n) + \sqrt{2\epsilon} M^{1/2} W_n \qquad n \in \mathbb{N},$$

where $M^{1/2}$ denotes the matrix square root of $M$. One natural choice is to take $\phi_0 = \text{argmax}_\phi \ln g(\phi)$, so the chain is initialized the posterior mode, and to take the preconditioning matrix $M = (\nabla^2 \ln g(\phi)|_{\phi=\phi_0})^{-1}$ to be the inverse Hessian at the posterior mode.

## 5.3 Laplace Approximation

The last two methods we described are instances of Markov chain Monte Carlo, which generate approximate samples from the target distribution by simulating a carefully constructed Markov chain. The technique described here instead explicitly approximates a potentially complicated posterior distribution by a Gaussian distribution. Samples from this simpler Gaussian

distribution can then serve as approximate samples from the posterior distribution of interest. [5] proposed using this method to approximate Thompson sampling in a display advertising problem with a logistic regression model of ad-click-through rates.

Let $g$ denote a probability density function over $\mathbb{R}^K$ from which we wish to sample. If $g$ is unimodal, and its log density $\ln g(\phi)$ is strictly concave around its mode $\overline{\phi}$, then $g(\phi) = e^{\ln g(\phi)}$ is sharply peaked around $\overline{\phi}$. It is therefore natural to consider approximating $g$ locally around its mode. A second-order Taylor approximation to the log-density gives

$$\ln g(\phi) \approx \ln g(\overline{\phi}) - \frac{1}{2}(\phi - \overline{\phi})^\top C(\phi - \overline{\phi}),$$

where

$$C = -\nabla^2 \ln g(\overline{\phi}).$$

As an approximation to the density $g$, we can then use

$$\tilde{g}(\phi) \propto e^{-\frac{1}{2}(\phi-\overline{\phi})^\top C(\phi-\overline{\phi})}.$$

This is proportional to the density of a Gaussian distribution with mean $\overline{\phi}$ and covariance $C^{-1}$, and hence

$$\tilde{g}(\phi) = \sqrt{|C/2\pi|} e^{-\frac{1}{2}(\phi-\overline{\phi})^\top C(\phi-\overline{\phi})}.$$

We refer to this as the Laplace approximation of $g$. Since there are efficient algorithms for generating normally-distributed samples, this offers a viable means to approximately sampling from $g$.

As an example, let us consider application of the Laplace approximation to Example 6. Bayes rule implies that the posterior density $f_{t-1}$ of $\theta$ satisfies

$$f_{t-1}(\theta) \propto f_0(\theta) \prod_{\tau=1}^{t-1} \left( \frac{1}{1 + \exp\left(\sum_{e \in x_\tau} \theta_e - M\right)} \right)^{y_\tau} \left( \frac{\exp\left(\sum_{e \in x_\tau} \theta_e - M\right)}{1 + \exp\left(\sum_{e \in x_\tau} \theta_e - M\right)} \right)^{1-y_\tau}.$$

The mode $\overline{\theta}$ can be efficiently computed via maximizing $f_{t-1}$, which is log-concave. An approximate posterior sample $\hat{\theta}$ is then drawn from a normal distribution with mean $\overline{\theta}$ and covariance matrix $(-\nabla^2 \ln f_{t-1}(\overline{\theta}))^{-1}$. To produce the computational results reported in Figure 8, we applied Newton's method with a backtracking line search to maximize $\ln f_{t-1}$.

Laplace approximations are well suited for Example 6 because the log-posterior density is strictly concave and its gradient and Hessian can be computed efficiently. Indeed, more broadly, Laplace approximations tend to be effective for posterior distributions with smooth densities that are sharply peaked around their mode. They tend to be computationally efficient when one can efficiently compute the posterior mode, and can efficiently form the Hessian of the log-posterior density.

The behavior of the Laplace approximation is not invariant to a substitution of variables, and it can sometimes be helpful to apply such a substitution. To illustrate this point, let us revisit the online shortest path problem of Example 5. For this problem, posterior distributions of $\theta$ are log-normal. However, these distributions are normal in $\phi$, where $\phi_e = \ln \theta_e$ for each edge $e \in E$. As such, if the Laplace approximation approach is applied to generate a sample $\hat{\phi}$ from the posterior distribution of $\phi$, the normal approximation is no longer an approximation, and, letting $\hat{\theta}_e = \exp(\hat{\phi}_e)$ for each $e \in E$, we obtain a sample $\hat{\theta}$ exactly from the posterior distribution of $\theta$. In this case, through a variable substitution, we can sample in a manner that makes the Laplace approximation exact. More broadly, for any given problem, it may be possible to introduce variable substitutions that enhance the efficacy of the Laplace approximation.

## 5.4 Bootstrapping

As a an alternative, we discuss an approach based on the statistical bootstrap, which accommodates even very complex densities. There are many versions of the bootstrap approach that can be used to approximately sample from a posterior distribution. For concreteness, we introduce a specific one that is suitable for examples we cover in this paper.

Like the Laplace approximation approach, our bootstrap method assumes that $\theta$ is drawn from a Euclidean space $\mathbb{R}^K$. Consider first a standard bootstrap method for evaluating the sampling distribution of the maximum likelihood estimate of $\theta$. The method generates a hypothetical history $\hat{\mathbb{H}}_{t-1} = ((\hat{x}_1, \hat{y}_1), \ldots, (\hat{x}_{t-1}, \hat{y}_{t-1}))$, which is made up of $t-1$ action-observation pairs, each sampled uniformly with replacement from $\mathbb{H}_{t-1}$. We then maximize the likelihood of $\theta$ under the hypothetical history, which for our shortest path recommendation problem is given by

$$\hat{L}_{t-1}(\theta) = \prod_{\tau=1}^{t-1} \left( \frac{1}{1 + \exp\left(\sum_{e \in \hat{x}_\tau} \theta_e - M\right)} \right)^{\hat{y}_\tau} \left( \frac{\exp\left(\sum_{e \in \hat{x}_\tau} \theta_e - M\right)}{1 + \exp\left(\sum_{e \in \hat{x}_\tau} \theta_e - M\right)} \right)^{1-\hat{y}_\tau}.$$

The randomness in the maximizer of $\hat{L}_{t-1}$ reflects the randomness in the sampling distribution of the maximum likelihood estimate. Unfortunately, this method does not take the agent's prior into account. A more severe issue is that it grossly underestimates the agent's real uncertainty in initial periods. The modification described here is intended to overcome these shortcomings in a simple way.

The method proceeds as follows. First, as before, we draw a hypothetical history $\hat{\mathbb{H}}_{t-1} = ((\hat{x}_1, \hat{y}_1), \ldots, (\hat{x}_{t-1}, \hat{y}_{t-1}))$, which is made up of $t-1$ action-observation pairs, each sampled uniformly with replacement from $\mathbb{H}_{t-1}$. Next, we draw a sample $\theta^0$ from the prior distribution $f_0$. Let $\Sigma$ denote the covariance matrix of the prior $f_0$. Finally, we solve the maximization problem

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^k}{\operatorname{argmax}} \; e^{-(\theta - \theta^0)^\top \Sigma (\theta - \theta^0)} \hat{L}_{t-1}(\theta)$$

and treat $\hat{\theta}$ as an approximate posterior sample. This can be viewed as maximizing a randomized approximation $\hat{f}_{t-1}$ to the posterior density, where $\hat{f}_{t-1}(\theta) \propto e^{-(\theta - \theta^0)^\top \Sigma (\theta - \theta^0)} \hat{L}_{t-1}(\theta)$ is what the posterior density would be if the prior were normal with mean $\theta^0$ and covariance matrix $\Sigma$, and the history of observations were $\hat{\mathbb{H}}_{t-1}$. When very little data has been gathered, the randomness in the samples mostly stems from the randomness in the prior sample $\theta_0$. This random prior sample encourages the agent to explore in early periods. When $t$ is large, so a lot of a data has been gathered, the likelihood typically overwhelms the prior sample and randomness in the samples mostly stems from the random selection of the history $\hat{\mathbb{H}}_{t-1}$.

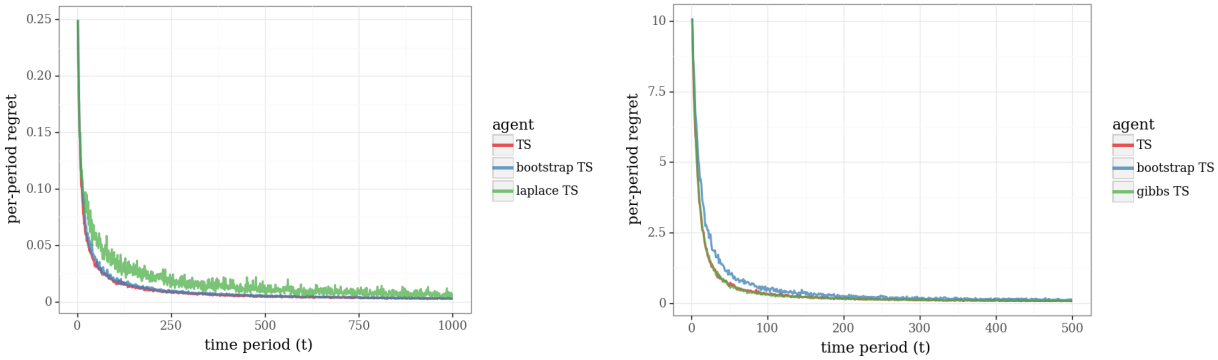
In the context of the shortest path recommendation problem, $\hat{f}_{t-1}(\theta)$ is log-concave and can therefore be efficiently maximized. Again, to produce our computational results reported in Figure 8, we applied Newton's method with a backtracking line search to maximize $\ln \hat{f}_{t-1}$. Even when it is not possible to efficiently maximize $\hat{f}_{t-1}$, however, the bootstrap approach can be applied with heuristic optimization methods that identify local or approximate maxima.

## 5.5 Sanity Checks

Figure 8 demonstrates that Laplace approximation and bootstrap approaches, when applied to the path recommendation problem, learn from binary feedback to improve performance over time. This may leave one wondering, however, whether exact Thompson sampling would offer substantially better performance. Since we do not have a tractable means of carrying out exact Thompson sampling for this problem, in this section, we apply our approximation methods to problems for which exact Thompson sampling is tractable. This enables comparisons between performance of exact and approximate methods.

Recall the three-armed beta-Bernoulli bandit problem for which results from application of greedy and Thompson sampling algorithms were reported in Figure 4(b). For this problem, components of $\theta$ are independent under posterior distributions, and as such, Gibbs sampling yields exact posterior samples. Hence, the performance of an approximate version that uses Gibbs sampling would be identical to that of exact Thompson sampling. Figure 9a plots results from applying Laplace approximation and bootstrap approaches. For this problem, both approximation methods offer performance that is qualitatively similar to exact Thompson sampling. We do see that Laplace sampling performs marginally worse than Bootstrapping in this setting.

Next, consider the online shortest path problem with correlated edge delays. Regret experienced by Thompson sampling applied to such a problem were reported in Figure 7a. As discussed in Section 5.3, applying the Laplace approximation approach with an appropriate variable substitution leads to the same results as exact Thompson sampling. Figure 9b compares those results to what is generated by Gibbs sampling and bootstrap approaches. Again, the approximation methods yield competitive results, although bootstrapping is marginally less effective than Gibbs sampling.




(a) Bernoulli bandit

(b) online shortest path

Figure 9: Regret of approximation methods versus exact Thompson sampling.

## 5.6 Incremental Implementation

For each of the three approximation methods we have discussed, the compute time required per time period grows as time progresses. This is because each past observation must be accessed to generate the next action. This differs from exact Thompson sampling algorithms we discussed earlier, which maintain parameters that encode a posterior distribution, and update these parameters over each time period based only on the most recent observation.

In order to keep the computational burden manageable, it can be important to consider incremental variants of our approximation methods. We refer to an algorithm as *incremental* if it operates with fixed rather than growing per-period compute time. There are many ways to design incremental variants of approximate posterior sampling algorithms we have presented. As concrete examples, we consider here particular incremental versions of Laplace approximation and bootstrap approaches.

For each time $t$, let $\ell_t(\theta)$ denote the likelihood of $y_t$ conditioned on $x_t$ and $\theta$. Hence, conditioned on $\mathbb{H}_{t-1}$, the posterior density satisfies

$$f_{t-1}(\theta) \propto f_0(\theta) \prod_{\tau=1}^{t-1} \ell_\tau(\theta).$$

Let $g_0(\theta) = \ln f_0(\theta)$ and $g_t(\theta) = \ln \ell_t(\theta)$ for $t > 0$. To identify the mode of $f_{t-1}$, it suffices to maximize $\sum_{\tau=0}^{t-1} g_\tau(\theta)$.

Consider an incremental version of the Laplace approximation. The algorithm maintains statistics $H_t$, and $\overline{\theta}_t$, initialized with $\overline{\theta}_0 = \text{argmax}_\theta\, g_0(\theta)$, and $H_0 = \nabla^2 g_0(\overline{\theta}_0)$, and updating according to

$$H_t = H_{t-1} + \nabla^2 g_t(\overline{\theta}_{t-1}),$$
$$\overline{\theta}_t = \overline{\theta}_{t-1} - H_t^{-1} \nabla g_t(\overline{\theta}_{t-1}).$$

This algorithm is a type of online newton method for computing the posterior mode $\overline{\theta}_{t-1}$ that maximizes $\sum_{\tau=0}^{t-1} g_\tau(\theta)$. Note that if each function $g_{t-1}$ is strictly concave and quadratic, as would be the case if the prior is normal and observations are linear in $\theta$ and perturbed only by normal noise, each pair $\overline{\theta}_{t-1}$ and $H_{t-1}^{-1}$ represents the mean and covariance matrix of $f_{t-1}$. More broadly, these iterates can be viewed as the mean and covariance matrix of a Gaussian approximation to the posterior, and used to generate an approximate posterior sample $\hat{\theta} \sim N(\overline{\theta}_{t-1}, H_{t-1}^{-1})$. It worth noting that for linear and generalized linear models, the matrix $\nabla^2 g_t(\overline{\theta}_{t-1})$ has rank one, and therefore $H_t^{-1} = (H_{t-1} + \nabla^2 g_t(\overline{\theta}_{t-1}))^{-1}$ can be updated incrementally using the Sherman-Woodbury-Morrison formula. This incremental version of the Laplace approximation is closely related to the notion of an extended Kalman filter, which has been explored in greater depth by Gómez-Uribe [26] as a means for incremental approximate Thompson sampling with exponential families of distributions.

Another approach involves incrementally updating each of an ensemble of models to behave like a sample from the posterior distribution. The posterior can be interpreted as a distribution of "statistically plausible" models, by which we mean models that are sufficiently consistent with prior beliefs and the history of observations. With this interpretation in mind, Thompson sampling can be thought of as randomly drawing from the range of statistically plausible models. *Ensemble sampling* aims to maintain, incrementally update, and sample from a finite set of such models. In the spirit of particle filtering, this set of models approximates the posterior distribution. The workings of ensemble sampling are in some ways more intricate than conventional uses of particle filtering, however, because interactions between the ensemble of models and selected actions can skew the distribution. *Ensemble sampling* is presented in more depth in [27], which draws inspiration from work on exploration in deep reinforcement learning [15].

There are multiple ways of generating suitable model ensembles. One builds on the aforementioned bootstrap method and involves fitting each model to a different bootstrap sample. To elaborate, consider maintaining $N$ models with parameters $(\overline{\theta}_t^n, H_t^n : n = 1, \ldots, N)$. Each set is initialized with $\overline{\theta}_0^n \sim g_0$, $H_0^n = \nabla g_0(\overline{\theta}_0^n)$, $d_0^n = 0$, and updated according to

$$H_t^n = H_{t-1}^n + z_t^n \nabla^2 g_t(\overline{\theta}_{t-1}^n),$$
$$\overline{\theta}_t^n = \overline{\theta}_{t-1}^n - z_t^n (H_t^n)^{-1} \nabla g_t(\overline{\theta}_{t-1}^n),$$

where each $z_t^n$ is an independent Poisson-distributed sample with mean one. Each $\overline{\theta}_t^n$ can be viewed as a random statistically plausible model, with randomness stemming from the initialization of $\overline{\theta}_0^n$ and the random weight $z_t^n$ placed on each observation. The variable, $z_\tau^n$ can loosely be interpreted as a number of replicas of the data sample $(x_\tau, y_\tau)$ placed in a hypothetical history $\hat{\mathbb{H}}_t^n$. Indeed, in a data set of size $t$, the number of replicas of a particular bootstrap data sample follows a Binomial$(t, 1/t)$ distribution, which is approximately Poisson(1) when $t$ is large. With this view, each $\overline{\theta}_t^n$ is effectively fit to a different data set $\hat{\mathbb{H}}_t^n$, distinguished by the random number of replicas assigned to each data sample. To generate an action $x_t$, $n$ is sampled uniformly from $\{1, \ldots, N\}$, and the action is chosen to maximize $\mathbb{E}[r_t | \theta = \overline{\theta}_{t-1}^n]$. Here, $\overline{\theta}_{t-1}^n$ serves as the approximate posterior sample. Note that the per-period compute time grows with $N$, which is an algorithm tuning parameter.

This bootstrap approach offers one mechanism for incrementally updating an ensemble of models. In Section 7.3, we will discuss another along with its application to active learning in neural networks.

# 6 Practical Modeling Considerations

Our narrative over previous sections has centered around a somewhat idealized view of Thompson sampling, which ignored the process of prior specification and assumed a simple model in which the system and set of feasible actions is constant over time and there is no side information on decision context. In this section, we provide greater perspective on the process of prior specification and on extensions of Thompson sampling that serve practical needs arising in some applications.

## 6.1 Prior Distribution Specification

The algorithms we have presented require as input a prior distribution over model parameters. The choice of prior can be important, so let us now discuss its role and how it might be selected. In designing an algorithm for an online decision problem, unless the value of $\theta$ were known with certainty, it would not make sense to optimize performance for a single value, because that could lead to poor performance for other plausible values. Instead, one might design the algorithm to perform well on average across a collection of possibilities. The prior can be thought of as a distribution over plausible values, and its choice directs the algorithm to perform well on average over random samples from the prior.

For a practical example of prior selection, let us revisit the banner ad placement problem introduced in Example 1. There are $K$ banner ads for a single product, with unknown click-through probabilities $(\theta_1, \ldots, \theta_K)$. Given a prior, Thompson sampling can learn to select the most successful ad. We could use a uniform or, equivalently, a beta$(1, 1)$ distribution over each $\theta_k$. However, if some values of $\theta_k$ are more likely than others, using a uniform prior sacrifices performance. In particular, this prior represents no understanding of the context, ignoring any useful knowledge from past experience. Taking knowledge into account reduces what must be learned and therefore reduces the time it takes for Thompson sampling to identify the most effective ads.

Suppose we have a data set collected from experience with previous products and their ads, each distinguished by stylistic features such as language, font, and background, together with accurate estimates of click-through probabilities. Let us consider an empirical approach to prior selection that leverages this data. First, partition past ads into $K$ sets, with each $k$th partition consisting of those with stylistic features most similar to the $k$th ad under current consideration. Figure 10 plots a hypothetical empirical cumulative distribution of click-through probabilities for ads in the $k$th set. It is then natural to consider as a prior a smoothed approximation of this distribution, such as the beta$(1, 100)$ distribution also plotted in Figure 10. Intuitively, this process assumes that click-through probabilities of past ads in set $k$ represent plausible values of $\theta_k$. The resulting prior is informative; among other things, it virtually rules out click-through probabilities greater than 0.05.

A careful choice of prior can improve learning performance. Figure 11 presents results from simulations of a three-armed Bernoulli bandit. Mean rewards of the three actions are sampled from beta$(1, 50)$, beta$(1, 100)$, and beta$(1, 200)$ distributions, respectively. Thompson sampling is applied with these as prior distributions and with a uniform prior distribution. We refer to the latter as a *misspecified prior* because it is not consistent with our understanding of the problem. A prior that is consistent in this sense is termed *coherent*. Each plot represents an average over ten thousand independent simulations, each with independently sampled mean rewards.

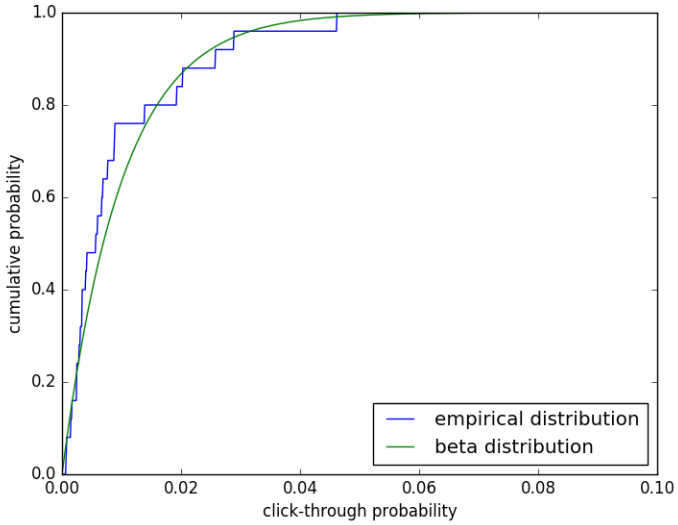
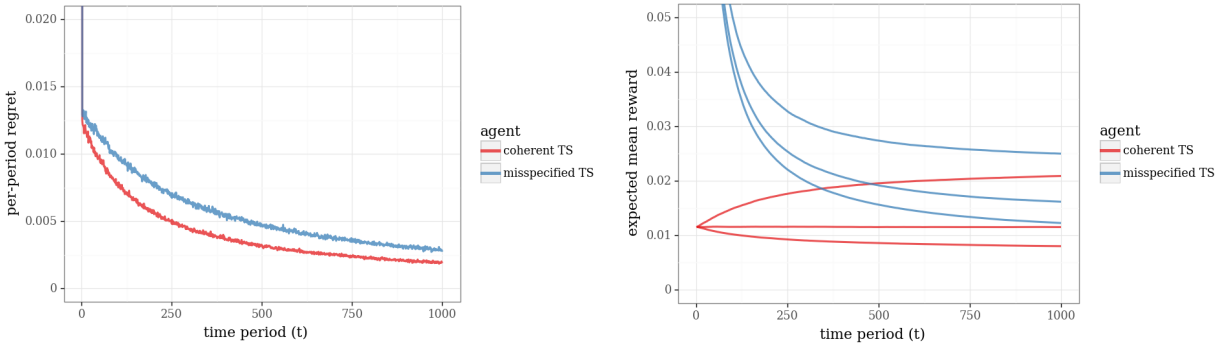



Figure 10: An empirical cumulative distribution and an approximating beta distribution.

Figure 11a plots expected regret, demonstrating that the misspecified prior increases regret. Figure 11a plots the evolution of the agent's mean reward conditional expectations. For each algorithm, there are three curves corresponding to the best, second-best, and worst actions, and they illustrate how starting with a misspecified prior delays learning.




(a) regret

(b) expected mean rewards

Figure 11: Comparison of Thompson sampling for the Bernoulli bandit problem with coherent versus misspecified priors.

## 6.2 Constraints, Context, and Caution

Though Algorithm 4, as we have presented it, treats a very general model, straightforward extensions accommodate even broader scope. One involves imposing **time-varying constraints** on the actions. In particular, there could be a sequence of admissible action sets $\mathcal{X}_t$ that constrain actions $x_t$. To motivate such an extension, consider our shortest path example. Here, on any given day, the drive to work may be constrained by announced road closures. If $\mathcal{X}_t$ does

not depend on $\theta$ except through possible dependence on the history of observations, Thompson sampling (Algorithm 4) remains an effective approach, with the only required modification being to constrain the maximization problem in Line 6.

Another extension of practical import addresses **contextual online decision problems**. In such problems, the response $y_t$ to action $x_t$ also depends on an independent random variable $z_t$ that the agent observes prior to making her decision. In such a setting, the conditional distribution of $y_t$ takes the form $p_\theta(\cdot|x_t, z_t)$. To motivate this, consider again the shortest path example, but with the agent observing a weather report $z_t$ from a news channel before selecting a path $x_t$. Weather may affect delays along different edges differently, and the agent can take this into account before initiating her trip. Contextual problems of this flavor can be addressed through augmenting the action space and introducing time-varying constraint sets. In particular, if we view $\tilde{x}_t = (x_t, z_t)$ as the action and constrain its choice to $\mathcal{X}_t = \{(x, z_t) : x \in \mathcal{X}\}$, where $\mathcal{X}$ is the set from which $x_t$ must be chosen, then it is straightforward to apply Thompson sampling to select actions $\tilde{x}_1, \tilde{x}_2, \ldots$. For the shortest path problem, this can be interpreted as allowing the agent to dictate both the weather report and the path to traverse, but constraining the agent to provide a weather report identical to the one observed through the news channel.

In some applications, it may be important to ensure that expected performance exceeds some prescribed baseline. This can be viewed as a level of **caution** against poor performance. For example, we might want each action applied to offer expected reward of at least some level $\underline{r}$. This can again be accomplished through constraining actions: in each $t$th time period, let the action set be $\mathcal{X}_t = \{x \in \mathcal{X} : \mathbb{E}[r_t|x_t = x] \geq \underline{r}\}$. Using such an action set ensures that expected average reward exceeds $\underline{r}$. When actions are related, an actions that is initially omitted from the set can later be included if what is learned through experiments with similar actions increases the agent's expectation of reward from the initially omitted action.

## 6.3 Nonstationary Systems

Problems we have considered involve model parameters $\theta$ that are constant over time. As Thompson sampling hones in on an optimal action, the frequency of exploratory actions converges to zero. In many practical applications, the agent faces a nonstationary system, which is more appropriately modeled by time-varying parameters $\theta_1, \theta_2, \ldots$, such that the response $y_t$ to action $x_t$ is generated according to $p_{\theta_t}(\cdot|x_t)$. In such contexts, the agent should never stop exploring, since it needs to track changes as the system drifts. With minor modification, Thompson sampling remains an effective approach so long as model parameters change little over durations that are sufficient to identify effective actions.

In principle, Thompson sampling could be applied to a broad range of problems where the parameters $\theta_1, \theta_2, ...$, evolve according to some stochastic process $p(\theta_t|\theta_1, ..., \theta_{t-1})$ by using techniques from filtering and sequential Monte Carlo to generate posterior samples. Instead we describe below some much simpler approaches to such problems.

One simple approach to addressing nonstationarity involves ignoring historical observations made beyond some number $\tau$ of time periods in the past. With such an approach, at each time $t$, the agent would produce a posterior distribution based on the prior and conditioned only on the most recent $\tau$ actions and observations. Model parameters are sampled from this distribution, and an action is selected to optimize the associated model. The agent never ceases to explore, since the degree to which the posterior distribution can concentrate is limited by the number of observations taken into account.

An alternative approach involves modeling evolution of a belief distribution in a manner that discounts the relevance of past observations and tracks a time-varying parameters $\theta_t$. We now consider such a model and a suitable modification of Thompson sampling. Let us start with the simple context of a Bernoulli bandit. Take the prior for each $k$th mean reward to be beta$(\alpha, \beta)$. Let the algorithm update parameters to identify the belief distribution of $\theta_t$ conditioned on the

history $\mathbb{H}_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$ according to

$$(\alpha_k, \beta_k) \leftarrow \begin{cases} \big((1-\gamma)\alpha_k + \gamma\overline{\alpha}, (1-\gamma)\beta_k + \gamma\overline{\beta}\big) & x_t \neq k \\ \big((1-\gamma)\alpha_k + \gamma\overline{\alpha} + r_t, (1-\gamma)\beta_k + \gamma\overline{\beta} + 1 - r_t\big) & x_t = k, \end{cases} \tag{6.1}$$

where $\gamma \in [0, 1]$ and $\overline{\alpha}_k, \overline{\beta}_k > 0$. This models a process for which the belief distribution converges to beta$(\overline{\alpha}_k, \overline{\beta}_k)$ in the absence of observations. Note that, in the absence of observations, if $\gamma > 0$ then $(\alpha_k, \beta_k)$ converges to $(\overline{\alpha}_k, \overline{\beta}_k)$. Intuitively, the process can be thought of as randomly perturbing model parameters in each time period, injecting uncertainty. The parameter $\gamma$ controls how quickly uncertainty is injected. At one extreme, when $\gamma = 0$, no uncertainty is injected. At the other extreme, $\gamma = 1$ and each $\theta_{t,k}$ is an independent beta$(\overline{\alpha}_k, \overline{\beta}_k)$-distributed process. A modified version of Algorithm 2 can be applied to this nonstationary Bernoulli bandit problem, the differences being in the additional arguments $\gamma$, $\overline{\alpha}$, and $\overline{\beta}$ , and the formula used to update distribution parameters.

The more general form of Thompson sampling presented in Algorithm 4 can be modified in an analogous manner. For concreteness, let us focus on the case where $\theta$ is restricted to a finite set; it is straightforward to extend things to infinite sets. The conditional distribution update in Algorithm 4 can be written as

$$p(u) \leftarrow \frac{p(u)q_u(y_t|x_t)}{\sum_v p(v)q_v(y_t|x_t)}.$$

To model nonstationary model parameters, we can use the following alternative:

$$p(u) \leftarrow \frac{\overline{p}^{\gamma}(u)p^{1-\gamma}(u)q_u(y_t|x_t)}{\sum_v \overline{p}^{\gamma}(v)p^{1-\gamma}(v)q_v(y_t|x_t)}.$$

This generalizes the formula provided earlier for the Bernoulli bandit case. Again, $\gamma$ controls the rate at which uncertainty is injected. The modified version of Algorithm 2, which we refer to as *nonstationary Thompson sampling*, takes $\gamma$ and $\overline{p}$ as additional arguments and replaces the distribution update formula.

Figure 12 illustrates potential benefits of nonstationary Thompson sampling when dealing with a nonstationairy Bernoulli bandit problem. In these simulations, belief distributions evolve according to Equation (6.1). The prior and stationary distributions are specified by $\alpha = \overline{\alpha} = \beta = \overline{\beta} = 1$. The decay rate is $\gamma = 0.01$. Each plotted point represents an average over 10,000 independent simulations. Regret here is defined by $\text{regret}_t(\theta_t) = \max_k \theta_{t,k} - \theta_{t,x_t}$. While nonstationary Thompson sampling updates its belief distribution in a manner consistent with the underlying system, Thompson sampling pretends that the success probabilities are constant over time and updates its beliefs accordingly. As the system drifts over time, Thompson sampling becomes less effective, while nonstationary Thompson sampling retains reasonable performance. Note, however, that due to nonstationarity, no algorithm can promise regret that vanishes with time.

# 7 Further Examples

As contexts for illustrating the workings of Thompson sampling, we have presented the Bernoulli bandit and variations of the online shortest path problem. To more broadly illustrate the scope of Thompson sampling and issues that arise in various kinds of applications, we present several additional examples in this section.

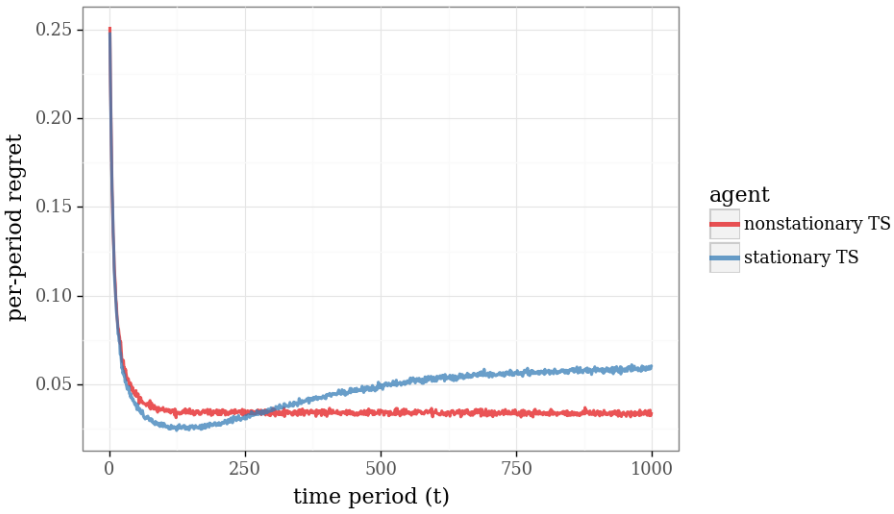



Figure 12: Comparison of Thompson sampling versus nonstationary Thompson sampling with a nonstationary Bernoulli bandit problem.

## 7.1 Product Assortment

Let us start with an assortment planning problem. Consider an agent who has an ample supply of each of $n$ different products, indexed by $i = 1, 2, \cdots, n$. The seller collects a profit of $p_i$ per unit sold of product type $i$. In each period, the agent has the option of offering a subset of the products for sale. Products may be substitutes or complements, and therefore the demand for a product may be influenced by the other products offered for sale in the same period. In order to maximize her profit, the agent needs to carefully select the optimal set of products to offer in each period. We can represent the agent's decision variable in each period as a vector $x \in \{0,1\}^n$ where $x_i = 1$ indicates that product $i$ is offered and $x_i = 0$ indicates that it is not. Upon offering an assortment containing product $i$ in some period, the agent observes a random lognormally distributed demand $d_i$. The mean of this lognormal distribution depends on the entire assortment $x$ and an uncertain matrix $\theta \in \mathbb{R}^{k \times k}$. In particular

$$\log(d_i) \mid \theta, x \sim N\left((\theta x)_i, \sigma^2\right)$$

where $\sigma^2$ is a known parameter that governs the level of idiosyncratic randomness in realized demand across periods. For any product $i$ contained in the assortment $x$,

$$(\theta x)_i = \theta_{ii} + \sum_{j \neq i} x_j \theta_{ij},$$

where $\theta_{ii}$ captures the demand rate for item $i$ if it were the sole product offered and each $\theta_{ij}$ captures the effect availability of product $j$ has on demand for product $i$. When an assortment $x$ is offered, the agent earns expected profit

$$\mathbb{E}\left[\sum_{i=1}^{n} p_i x_i d_i \mid \theta, x\right] = \sum_{i=1}^{n} p_i x_i e^{(\theta x)_i + \frac{\sigma^2}{2}}. \tag{7.1}$$

If $\theta$ were known, the agent would always select the assortment $x$ that maximizes her expected profit in (7.1). However, when $\theta$ is unknown, the agent needs to learn to maximize profit by exploring different assortments and observing the realized demands.

Thompson sampling can be adopted as a computationally efficient solution to this problem. We assume the agent begins with a multivariate Gaussian prior over $\theta$. Due to conjugacy properties of normal and lognormal distributions, the posterior distribution of $\theta$ remains normal after any number of periods. At the beginning of each $t$'th period, the Thompson sampling algorithm draws a sample $\hat{\theta}_t$ from this normal posterior distribution. Then, the agent selects an assortment that would maximize her expected profit in period $t$ if the sampled $\hat{\theta}_t$ were indeed the true parameter.

As in Examples 4 and 5, the mean and covariance matrix of the posterior distribution of $\theta$ can be updated in closed form. However, because $\theta$ is a matrix rather than a vector, the explicit form of the update is more complicated. To describe the update rule, we first introduce $\bar{\theta}$ as the vectorized version of $\theta$ which is generated by stacking the columns of $\theta$ on top of each other. Let $x$ be the assortment selected in a period, $i_1, i_2, \cdots, i_k$ denote the the products included in this assortment (i.e., $\text{supp}(x) = \{i_1, i_2, \cdots, i_k\}$) and $z \in \mathbb{R}^k$ be defined element-wise as

$$z_j = \ln d_{i_j}, \quad j = 1, 2, \cdots, k.$$

Let $S$ be a $k \times n$ "selection matrix" where $S_{j,i_j} = 1$ for $j = 1, 2, \cdots, k$ and all its other elements are 0. Also, define

$$W = x^\top \otimes S,$$

where $\otimes$ denotes the Kronecker product of matrices. At the end of current period, the posterior mean $\mu$ and covariance matrix $\Sigma$ of $\bar{\theta}$ are updated according to the following rules:

$$\mu \leftarrow \left(\Sigma^{-1} + \frac{1}{\sigma^2} W^\top W\right)^{-1} \left(\Sigma^{-1}\mu + \frac{1}{\sigma^2} W^\top z\right), \quad \Sigma \leftarrow \left(\Sigma^{-1} + \frac{1}{\sigma^2} W^\top W\right)^{-1}.$$

To investigate the performance of Thompson sampling in this problem, we simulated a scenario with $n = 6$ and $\sigma^2 = 0.04$. We take the profit associated to each product $i$ to be $p_i = 1/6$. As the prior distribution, we assumed that each element of $\theta$ is independently normally distributed with mean 0, the diagonal elements have a variance of 1, and the off-diagonal elements have a variance of 0.2. To understand this choice, recall the impact of diagonal and off-diagonal elements of $\theta$. The diagonal element $\theta_{ii}$ controls the mean demand when only product $i$ is available, and reflects the inherent quality or popularity of that item. The off-diagonal element $\theta_{ij}$ captures the influence availability of product $j$ has on mean demand for product $i$. Our choice of prior covariance encodes that the dominant effect on demand of a product is likely its own characteristics, rather than its interaction with any single other product. Figure 13 presents the performance of different learning algorithms in this problem. In addition to Thompson sampling, we have simulated the greedy and $\epsilon$-greedy algorithms for various values of $\epsilon$. We found that $\epsilon = 0.07$ provides the best performance for $\epsilon$-greedy in this problem.

As illustrated by this figure, the greedy algorithm performs poorly in this problem while $\epsilon$-greedy presents a much better performance. We found that the performance of $\epsilon$-greedy can be improved by using an annealing $\epsilon$ of $\frac{m}{m+t}$ at each period $t$. Our simulations suggest using $m = 9$ for the best performance in this problem. Figure 13 shows that Thompson sampling outperforms both variations of $\epsilon$–greedy in this problem.

## 7.2 Cascading Recommendations

We consider an online recommendation problem in which an agent learns to recommend a desirable list of *items* to a *user*. As a concrete example, the agent could be a search engine and the items could be web pages. We consider formulating this problem as a *cascading bandit*, in which user selections are governed by a *cascade model*, as is commonly used in the fields of information retrieval and online advertising [28].

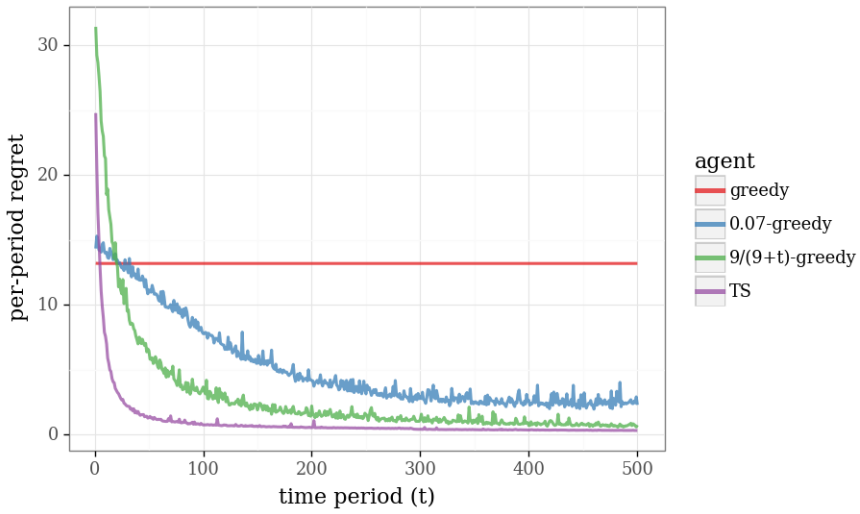



Figure 13: Regret experienced by different learning algorithms applied to product assortment problem.

A *cascading bandit* is represented by a tuple $(E, J, \theta)$, where $E = \{1, \ldots, K\}$ is the item set, $J \leq K$ is the number of recommended items in each period, and the model parameters $\theta \in [0,1]^K$ encode the item *attraction probabilities*. The agent knows the item set $E$ and the display size $J$, but must learn the user preferences encoded in $\theta$ through sequential interactions. We study the agent's performance over $T$ interactions.

During each time $t$, the agent chooses an ordered list $x_t = (x_{t,1}, \ldots, x_{t,J})$ of $J$ distinct items from $E$ based on its prior information and past observations. The list is then presented to the user, who either clicks on a single item from the list, or departs without clicking. The user's choices depend on the item attraction probabilities, their ordering in the vector $x_t$, and idiosyncratic randomness associated with the current time period. In particular, the user's choice process can be described as follows. Let $w_t \in \{0,1\}^K$ be a random binary vector generated by independently sampling each $k$th element $w_{t,k}$ from the Bernoulli distribution $\text{Bern}(\theta_k)$, where $\theta_k$ is the $k$th element of $\theta$. Item $k$ is *attractive* for the user at time $t$ if and only if $w_{t,k} = 1$. Following the cascade model, the user *views* items in the recommended list $x_t$ individually in order, stopping at and clicking the first that is attractive. Specifically, the $j$th item in $x_t$, is viewed:

- If $x_{t,j}$ is attractive at time $t$, then the user clicks $x_{t,j}$ and leaves.
- If $x_{t,j}$ is not attractive at time $t$ and $j < J$, then the user continues to examine $x_{t,j+1}$.
- If $x_{t,j}$ is not attractive at time $t$ and $j = J$, then the user leaves without a click.

Note that item $x_{t,1}$ is always examined, and there is at most one click at time $t$. Let $y_t = \text{argmin}\{1 \leq j \leq J : w_{t,x_{t,j}} = 1\}$ denote the user's choice, with the the convention that $\text{argmin}\, \emptyset = \infty$. Thus, $y_t = \infty$ when the user leaves without clicking.

The agent observes the user's choice $y_t$ and associates this with a reward $r_t = r(y_t) = \mathbf{1}\{y_t \leq J\}$ indicating whether the user clicked on some item. Note that based on $y_t$, the agent can update its estimates of attraction probabilities of all items $a_t^1, \ldots, a_t^{\min\{y_t, J\}}$ examined by the user. In particular, upon seeing the click $y_t < \infty$ the agent infers that item was attractive and each item presented before $y_t$ was unattractive. When no click was observed, the agent infers that every item in $x_t$ was unattractive to the user at time $t$.

For each ordered lists $x = (x_1, \ldots, x_J)$ and $\theta' \in [0,1]^K$, let

$$h(x, \theta') = 1 - \prod_{j=1}^{J} \left[1 - \theta'_{x_j}\right],$$

where $\theta'_{x_j}$ is the $x_j$-th element of $\theta'$. Note that $r_t = h(x_t, w_t)$, and the expected reward at time $t$ is $\mathbb{E}[r_t | x_t, \theta] = h(x_t, \theta)$. The optimal solution $x^* \in \text{argmax}_{x:|x|=J} h(x, \theta)$ consists of the $J$ items with highest attraction probabilities. The per-period regret of the cascading bandit is defined as

$$\text{regret}_t(\theta) = h(x^*, \theta) - h(x_t, \theta).$$

**Algorithm 5** CascadeUCB

Initialize $\alpha_k$ and $\beta_k$ $\forall k \in E$
**for** $t = 1, 2, \ldots$ **do**
  #compute itemwise UCBs:
  **for** $k = 1, \ldots, K$ **do**
    Compute UCB $\text{U}_t(k)$
  **end for**

  #select and apply action:
  $x_t \leftarrow \text{argmax}_{x:|x|=J} h(x, \text{U}_t)$
  Apply $x_t$ and observe $y_t$ and $r_t$

  #update sufficient statistics:
  **for** $j = 1, \ldots, \min\{y_t, J\}$ **do**
    $\alpha_{x_{t,j}} \leftarrow \alpha_{x_{t,j}} + \mathbf{1}(j = y_t)$
    $\beta_{x_{t,j}} \leftarrow \beta_{x_{t,j}} + \mathbf{1}(j < y_t)$
  **end for**
**end for**

**Algorithm 6** CascadeTS

Initialize $\alpha_k$ and $\beta_k$ $\forall k \in E$
**for** $t = 1, 2, \ldots$ **do**
  #sample model:
  **for** $k = 1, \ldots, K$ **do**
    Sample $\hat{\theta}_k \sim \text{Beta}(\alpha_k, \beta_k)$
  **end for**

  #select and apply action:
  $x_t \leftarrow \text{argmax}_{x:|x|=J} h(x, \hat{\theta})$
  Apply $x_t$ and observe $y_t$ and $r_t$

  #update posterior:
  **for** $j = 1, \ldots, \min\{y_t, J\}$ **do**
    $\alpha_{x_{t,j}} \leftarrow \alpha_{x_{t,j}} + \mathbf{1}(j = y_t)$
    $\beta_{x_{t,j}} \leftarrow \beta_{x_{t,j}} + \mathbf{1}(j < y_t)$
  **end for**
**end for**

Kveton et al. [29] proposed learning algorithms for cascading bandits based on itemwise UCB estimates. CascadeUCB (Algorithm 5) is practical variant that allows for specification of prior parameters $(\alpha, \beta)$ that guide early behavior of the algorithm. CascadeUCB computes a UCB $\text{U}_t(k)$ for each item $k \in E$ and then chooses the a list that maximizes $h(\cdot, \text{U}_t)$, which represents an upper confidence bound on the list attraction probability. The list $x_t$ can be efficiently generated by choosing the $J$ items with highest UCBs. Upon observing the user's response, the algorithm updates the sufficient statistics $(\alpha, \beta)$, which count clicks and views.

CascadeTS (Algorithm 6) is a Thompson sampling algorithm for cascading bandits. CascadeTS operates in a manner similar to CascadeUCB except that $x_t$ is computed based on the sampled attraction probabilities $\hat{\theta}$, rather than the itemwise UCBs $\text{U}_t$.

A standard form of UCB, known as UCB1, which is considered and analyzed in the context of cascading bandits in [29], takes the form

$$\text{U}_t(k) = \frac{\alpha_k}{\alpha_k + \beta_k} + \sqrt{\frac{1.5 \log(t)}{\alpha_k + \beta_k}}, \quad \forall k \in E.$$

Note that $\alpha_k/(\alpha_k + \beta_k)$ represents the expected value of the click probability $\theta_k$, while the second term represents an optimistic boost that encourages exploration. As the observations accumulate, the denominator $\sqrt{\alpha_k + \beta_k}$ grows, reducing the degree of optimism.

Figure 14 presents results from applying CascadeTS and CascadeUCB based on UCB1. These results are generated by randomly sampling 1000 cascading bandit instances, $K = 1000$ and $J = 100$, in each case sampling each attraction probability $\theta_k$ independently from Beta$(1, 40)$. For each instance, CascadeUCB and CascadeTS are applied over 20000 time periods, initialized with $(\alpha_k, \beta_k) = (1, 40)$. The plots are of per-period regrets averaged over the 1000 simulations.

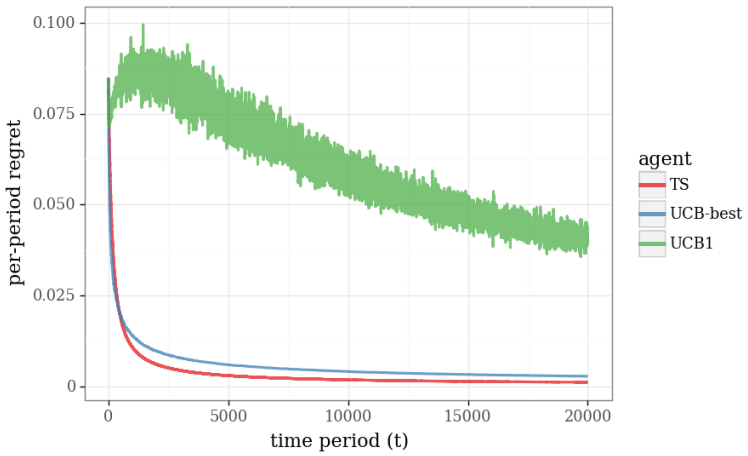



Figure 14: Comparison of CascadeTS and CascadeUCB with $K = 1000$ items and $J = 100$ recommendations per period.

The results demonstrate that TS far outperforms this version of CascadeUCB. Why? An obvious reason is that $h(x, \mathrm{U}_t)$ is far too optimistic. In particular, $h(x, \mathrm{U}_t)$ represents the probability of a click if *every* item in $x$ *simultaneously* takes on the largest attraction probability that is statistically plausible. However, the $w_{t,k}$'s are statistically independent across items and the agent is unlikely to have substantially under-estimated the attraction probability of every item in $x$. As such, $h(x, \mathrm{U}_t)$ tends to be far too optimistic. CascadeTS, on the other hand, samples $\hat{\theta}_k$'s independently across items. While any sample $\hat{\theta}_k$ might deviate substantially from its mean, it is unlikely that the sampled attraction probability of every item in $x$ greatly exceeds its mean. As such, the variability in $h(x, \hat{\theta})$ provides a much more accurate reflection of the magnitude of uncertainty.

One can address excess optimism by tuning the degree of optimism associated with UCB1. In particular, consider a UCB of the form

$$\mathrm{U}_t(k) = \frac{\alpha_k}{\alpha_k + \beta_k} + c\sqrt{\frac{1.5\log(t)}{\alpha_k + \beta_k}}, \quad \forall k \in E,$$

with the parameter $c$, which indicates the degree of optimism, selected through simulations to optimize performance. The plot labeled "UCB-tuned" in Figure 14 illustrates performance with $c = 0.05$, which approximately minimizes cumulative regret over 20,000 time periods. Table 1 provides the cumulative regret averages for each algorithm and range of degrees of optimism $c$, along with one standard deviation confidence interval boundaries. It is interesting that even after being tuned to the specific problem and horizon, the performance of CascadeUCB falls short of Cascade TS. A likely source of loss stems from the shape of confidence sets used by CascadeUCB. Note that the algorithm uses hyper-rectangular confidence sets, since the set of statistically plausible attraction probability vectors is characterized by a Cartesian product item-level confidence intervals. However, the Bayesian central limit theorem suggests that

"ellipsoidal" confidence sets offer a more suitable choice. Specifically, as data is gathered the posterior distribution over $\theta$ can be well approximated by a multivariate Gaussian, for which level sets are ellipsoidal. Losses due to the use of hyper-rectangular confidence sets have been studied through regret analysis in [30] and through simple analytic examples in [31].

Table 1: Comparison of CascadeTS and CascadeUCB with $K = 1000$ items and $J = 100$ recommendations per period over a range of optimism parameters.

| algorithm | degree of optimism | cumulative regret |
|---|---|---|
| TS | NA | $\mathbf{71.548 \pm 0.395}$ |
| UCB | 0 (greedy) | $135.410 \pm 0.820$ |
| | 0.0001 | $134.768 \pm 0.807$ |
| | 0.001 | $133.906 \pm 0.803$ |
| | 0.005 | $131.737 \pm 0.756$ |
| | 0.01 | $127.454 \pm 0.726$ |
| | 0.05 | $\mathbf{115.345 \pm 0.605}$ |
| | 0.1 | $125.351 \pm 0.636$ |
| | 0.2 | $200.221 \pm 1.069$ |
| | 0.3 | $324.586 \pm 1.644$ |
| | 0.5 | $615.070 \pm 2.705$ |
| | 0.75 | $954.670 \pm 3.647$ |
| | 1 (UCB1) | $1222.757 \pm 4.032$ |

It is worth noting that tuned versions of CascadeUCB do sometimes perform as well or better than CascadeTS. Table 2 and Figure 15 illustrates an example of this. The setting is identical to that used to generate the results of 14, except that $K = 50$ and $J = 10$, and cumulative regret is approximately optimized with $c = 0.1$. CascadeUCB outperforms CascadeTS. This qualitative difference from the case of $K = 1000$ and $J = 100$ is likely due to the fact that hyper-rectangular sets offer poorer approximations of ellipsoids as the dimension increases. This phenomenon and its impact on regret aligns with theoretical results of [30]. That said, CascadeUCB is somewhat advantaged in this comparison because it is tuned specifically for the setting and time horizon.

Results presented so far initialize the algorithms with coherent priors. To illustrate both robustness and potentially complex behaviors of these algorithms, we present in Table 3 and Figure 16 results generated when the algorithms are initialized with uniform priors ($\alpha_k = \beta_k = 1$). Aside from these priors, the setting is identical to that used to generate Table 1 and Figure 14. CascadeTS far outperforms the nominal version of CascadeUCB and does not fare too much worse than CascadeTS initialized with a coherent prior. A puzzling fact, however, is that the tuned version of CascadeUCB is optimized by $c = 0$, which corresponds to the greedy algorithm. It could be that the greedy algorithm performs well here because the particular choice of misspecified prior induces a suitable kind of optimistic behavior. More broadly, this illustrates how ad hoc tuning can substantially improve the behavior of an algorithm in any given simulated context. However, Thompson sampling appears to operate robustly across a broad range of problems even when applied in a principled manner that does not require context-specific tuning.

Table 2: Comparison of CascadeTS and CascadeUCB with $K = 50$ items and $J = 10$ recommendations per period over a range of optimism parameters.

| algorithm | degree of optimism | cumulative regret |
|---|---|---|
| TS | NA | $\mathbf{183.391 \pm 1.050}$ |
| UCB | 0 (greedy) | $333.650 \pm 4.714$ |
| | 0.0001 | $325.448 \pm 4.618$ |
| | 0.001 | $325.007 \pm 4.648$ |
| | 0.005 | $309.303 \pm 4.408$ |
| | 0.01 | $297.496 \pm 4.257$ |
| | 0.05 | $205.882 \pm 2.861$ |
| | 0.1 | $\mathbf{159.626 \pm 1.695}$ |
| | 0.2 | $190.457 \pm 1.099$ |
| | 0.3 | $260.022 \pm 1.184$ |
| | 0.5 | $406.080 \pm 1.630$ |
| | 0.75 | $570.547 \pm 2.189$ |
| | 1 (UCB1) | $711.117 \pm 2.627$ |

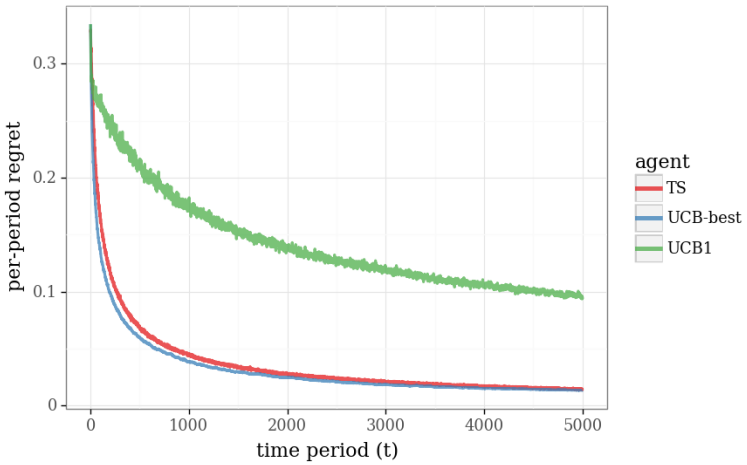



Figure 15: Comparison of CascadeTS and CascadeUCB with $K = 50$ items and $J = 10$ recommendations per period.

Table 3: Comparison of CascadeTS and CascadeUCB, both initialized with uniform priors, with $K = 1000$ items and $J = 100$ recommendations per period over a range of optimism parameters.

| algorithm | degree of optimism | cumulative regret |
|---|---|---|
| TS | NA | $\mathbf{82.458 \pm 0.449}$ |
| UCB | 0 (greedy) | $\mathbf{101.556 \pm 0.477}$ |
| | 0.0001 | $104.910 \pm 0.474$ |
| | 0.001 | $104.572 \pm 0.459$ |
| | 0.005 | $105.528 \pm 0.474$ |
| | 0.01 | $108.136 \pm 0.483$ |
| | 0.05 | $131.654 \pm 0.559$ |
| | 0.1 | $172.275 \pm 0.709$ |
| | 0.2 | $283.607 \pm 1.149$ |
| | 0.3 | $417.461 \pm 1.629$ |
| | 0.5 | $702.015 \pm 2.583$ |
| | 0.75 | $1024.434 \pm 3.390$ |
| | 1 (UCB1) | $1277.005 \pm 3.812$ |

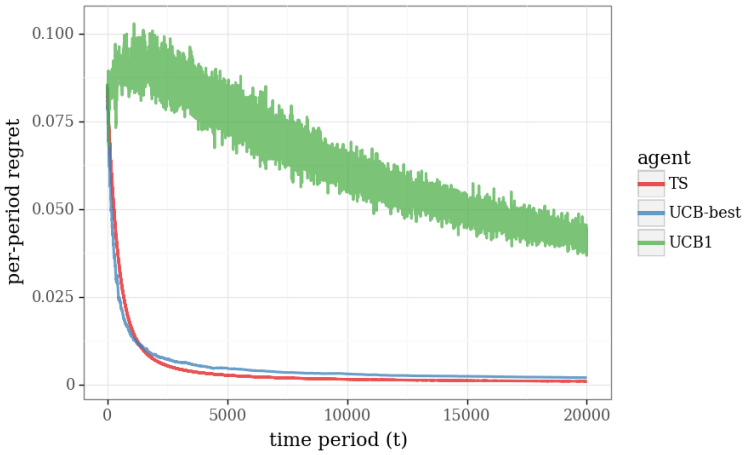



Figure 16: Comparison of CascadeTS and CascadeUCB, both initialized with uniform priors, with $K = 1000$ items and $J = 100$ recommendations per period.

## 7.3 Active Learning in Neural Networks

Neural networks are widely used in supervised learning, where given an existing set of predictor-response data pairs, the objective is to produce a model that generalizes to accurately predict future responses conditioned on associated predictors. They are also increasingly being used to guide actions ranging from recommendations to robotic maneuvers. Active learning is called for to close the loop by generating actions that do not solely maximize immediate performance but also probe the environment to generate data that accelerates learning. Thompson sampling offers a useful principle upon which such active learning algorithms can be developed.

With neural networks or other complex model classes, computing the posterior distribution over models becomes intractable. Approximations are called for, and incremental updating is essential because fitting a neural network is a computationally intensive task in its own right. In such contexts, ensemble sampling offers a viable approach [27]. In Section 5.6, we introduced a particular mechanism for ensemble sampling based on the bootstrap. In this section, we consider an alternative version of ensemble sampling and present results from [27] that demonstrate its application to active learning in neural networks.

To motivate our algorithm, let us begin by discussing how it can be applied to the linear bandit problem.

> **Example 7** (Linear Bandit) Let $\theta$ be drawn from $\mathbb{R}^M$ and distributed according to a $N(\mu_0, \Sigma_0)$ prior. There is a set of $K$ actions $\mathcal{X} \subseteq \mathbb{R}^M$. At each time $t = 1, \ldots, T$, an action $x_t \in \mathcal{X}$ is selected, after which a reward $r_t = y_t = \theta^\top x_t + w_t$ is observed, where $w_t \sim N(0, \sigma_w^2)$.

In this context, ensemble sampling is unwarranted, since exact Bayesian inference can be carried out efficiently via Kalman filtering. Nevertheless, the linear bandit offers a simple setting for explaining the workings of an ensemble sampling algorithm.

Consider maintaining a covariance matrix updated according to

$$\Sigma_{t+1} = \left(\Sigma_t^{-1} + x_t x_t^\top / \sigma_w^2\right)^{-1},$$

and $N$ models $\overline{\theta}_t^1, \ldots, \overline{\theta}_t^N$, initialized with $\overline{\theta}_1^1, \ldots, \overline{\theta}_1^N$ each drawn independently from $N(\mu_0, \Sigma_0)$ and updated incrementally according to

$$\overline{\theta}_{t+1}^n = \Sigma_{t+1}\left(\Sigma_t^{-1}\overline{\theta}_t^n + x_t(y_t + \tilde{w}_t^n)/\sigma_w^2\right),$$

for $n = 1, \ldots, N$, where $(\tilde{w}_t^n : t = 1, \ldots, T, n = 1, \ldots, N)$ are independent $N(0, \sigma_w^2)$ random samples drawn by the updating algorithm. It is easy to show that the resulting parameter vectors satisfy

$$\overline{\theta}_t^n = \arg\min_\nu \left(\frac{1}{\sigma_w^2}\sum_{\tau=1}^{t-1}(y_\tau + \tilde{w}_\tau^n - x_\tau^\top \nu)^2 + (\nu - \overline{\theta}_1^n)^\top \Sigma_0^{-1}(\nu - \overline{\theta}_1^n)\right).$$

Thich admits an intuitive interpretation: each $\overline{\theta}_t^n$ is a model fit to a randomly perturbed prior and randomly perturbed observations. As established in [27], for any deterministic sequence $x_1, \ldots, x_{t-1}$, conditioned on the history, the models $\overline{\theta}_t^1, \ldots, \overline{\theta}_t^N$ are independent and identically distributed according to the posterior distribution of $\theta$. In this sense, the ensemble approximates the posterior.

The ensemble sampling algorithm we have described for the linear bandit problem motivates an analogous approach for the following neural network model.

> **Example 8** (Neural Network) Let $g_\theta : \mathbb{R}^M \mapsto \mathbb{R}^K$ denote a mapping induced by a neural network with weights $\theta$. Suppose there are $K$ actions $\mathcal{X} \subseteq \mathbb{R}^M$, which

serve as inputs to the neural network, and the goal is to select inputs that yield desirable outputs. At each time $t = 1, \ldots, T$, an action $x_t \in \mathcal{X}$ is selected, after which $y_t = g_\theta(x_t) + w_t$ is observed, where $w_t \sim N(0, \sigma_w^2 I)$. A reward $r_t = r(y_t)$ is associated with each observation. Let $\theta$ be distributed according to a $N(\mu_0, \Sigma_0)$ prior. The idea here is that data pairs $(x_t, y_t)$ can be used to fit a neural network model, while actions are selected to trade off between generating data pairs that reduce uncertainty in neural network weights and those that offer desirable immediate outcomes.

Consider an ensemble sampling algorithm that once again begins with $N$ independent models with connection weights $\overline{\theta}_1^1, \ldots, \overline{\theta}_1^N$ sampled from a $N(\mu_0, \Sigma_0)$ prior. It could be natural here to let $\mu_0 = 0$ and $\Sigma_0 = \sigma_0^2 I$ for some variance $\sigma_0^2$ chosen so that the range of probable models spans plausible outcomes. To incrementally update parameters, at each time $t$, each $n$th model applies some number of stochastic gradient descent iterations to reduce a loss function of the form

$$\mathcal{L}_t(\nu) = \frac{1}{\sigma_w^2} \sum_{\tau=1}^{t-1} (y_\tau + \tilde{w}_\tau^n - g_\nu(x_\tau))^2 + (\nu - \overline{\theta}_1^n)^\top \Sigma_0^{-1} (\nu - \overline{\theta}_1^n).$$

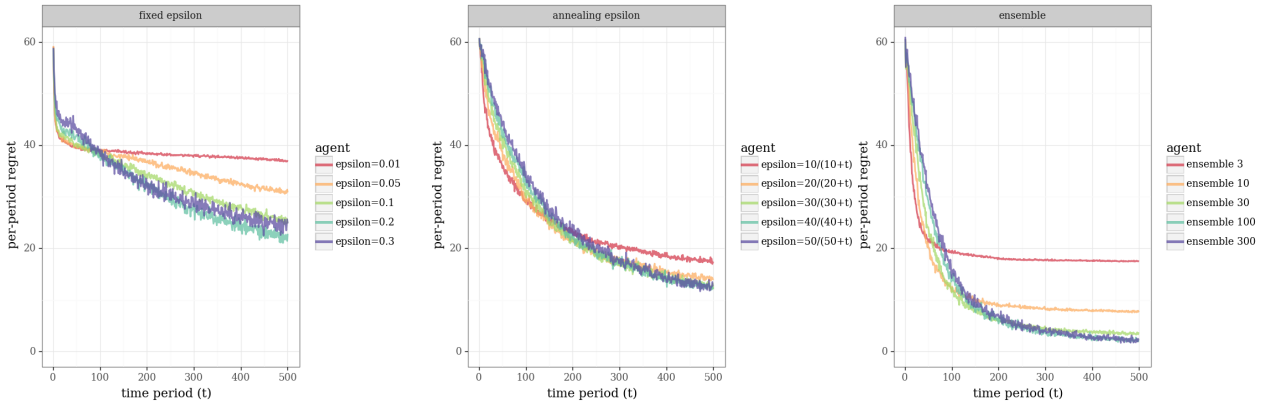
Figure 17 present results from simulations involving a two-layer neural network, with a set of $K$ actions, $\mathcal{X} \subseteq \mathbb{R}^M$. The weights of the neural network, which we denote by $w_1 \in \mathbb{R}^{D \times N}$ and $w_2 \in \mathbb{R}^D$, are each drawn from $N(0, \lambda)$. Let $\theta \equiv (w_1, w_2)$. The mean reward of an action $x \in \mathcal{X}$ is given by $g_\theta(x) = w_2^\top \max(0, w_1 a)$. At each time step, we select an action $x_t \in \mathcal{X}$ and observe reward $y_t = g_\theta(x_t) + z_t$, where $z_t \sim N(0, \sigma_z^2)$. We used $M = 100$ for the input dimension, $D = 50$ for the dimension of the hidden layer, number of actions $K = 100$, prior variance $\lambda = 1$, and noise variance $\sigma_z^2 = 100$. Each component of each action vector is sampled uniformly from $[-1, 1]$, except for the last component, which is set to 1 to model a constant offset. All results are averaged over 100 realizations.

In our application of the ensemble sampling algorithm we have described, to facilitate gradient flow, we use leaky rectified linear units of the form $\max(0.01x, x)$ during training, though the target neural network is made up of regular rectified linear units as indicated above. In our simulations, each update was carried out with 5 stochastic gradient steps, with a learning rate of $10^{-3}$ and a minibatch size of 64.

Figure 17 illustrates the performance of several learning algorithms with an underlying neural network. Figure 17a demonstrates the performance of an $\epsilon$-greedy strategy across various levels of $\epsilon$. We find that we are able to improve performance with an annealing schedule $\epsilon = \frac{k}{k+t}$ (Figure 17b). However, we find that an ensemble sampling strategy outperforms even the best tuned $\epsilon$-schedules (Figure 17c). Further, we see that ensemble sampling strategy can perform well with remarkably few members of this ensemble. Ensemble sampling with fewer members leads to a greedier strategy, which can perform better for shorter horizons, but is prone to premature and suboptimal convergence compared to true Thompson sampling [27]. In this problem, using an ensemble of as few as 30 members provides very good performance.

## 7.4 Reinforcement Learning in Markov Decision Processes

Reinforcement learning (RL) extends upon the contextual online decision problems covered in Section 6.2 to allow for delayed feedback and long term consequences [32, 33]. Concretely (using the notation of Section 6.2) the response $y_t$ to the action $x_t$ depends on a context $z_t$; but we no longer assume that the evolution of the context $z_{t+1}$ is independent of the action $x_t$. As such, the action $x_t$ may affect not only the reward $r(y_t)$ but also, through the effect upon the context $z_{t+1}$, the rewards earned in future timesteps $\{r(y_{t'})\}_{t'>t}$. As a motivating example, consider a problem of sequentially recommending products $x_t$ where the customer response $y_t$ is informed not only by the quality of the product, but also the history of past recommendations. The




(a) Fixed $\epsilon$-greedy. (b) Annealing $\epsilon$-greedy. (c) Ensemble TS.

Figure 17: Bandit learning with an underlying neural network.

evolution of the context $z_{t+1}$–which captures relevant information about the current customer available at time $t+1$–is then directly affected by the customer response $y_t$; if a customer watched 'The Godfather' and loved it, then chances are probably higher they may enjoy 'The Godfather 2'.

Maximizing cumulative rewards in a problem where decision have long term consequences can require planning with regards to future rewards, rather than optimizing each timestep myopically. Similarly, efficient exploration in these domains can require planning with regards to the potential for informative observations in future timesteps, rather than myopically considering the information gained over a single timestep. This sophisticated form of temporally-extended exploration, which can be absolutely critical for effective performance, is sometimes called *deep exploration* [34].

The Thompson sampling principle can also be applied successfully to reinforcement learning in finite horizon Markov decision processes (MDPs) [35]. However, we also highlight that special care must be taken with respect to the notion of 'timestep or 'period within Thompson sampling to preserve deep exploration.

An episodic finite horizon MDP $M = (\mathcal{S}, \mathcal{A}, R^M, P^M, H, \rho)$ is a type of online decision problem that proceeds in distinct episodes, each with $H$ timesteps within them. $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space and $H$ is the length of the episode. At the start of each episode the initial state $s_0$ is drawn from the distribution $\rho$. At each timestep $h = 0, .., H-1$ within an episode the agent observes state $s_h \in \mathcal{S}$, selects action $a_h \in \mathcal{A}$, receives a reward $r_h \sim R^M(s_h, a_h)$ and transitions to a new state $s_{h+1} \sim P^M(s_h, a_h)$. A policy $\mu$ is a function mapping each state $s \in \mathcal{S}$ and timestep $h = 0, .., H-1$ to an action $a \in \mathcal{A}$. The value function $V^M_{\mu,h}(s) = \mathbb{E}[\sum_{j=h}^{H-1} r_j(s_j, \mu(s_j, j)) \mid s_h = s]$ encodes the expected reward accumulated under $\mu$ in the remainder of the episode when starting from state $s$ and timestep $h$. Finite horizon MDPs allow for long term consequences through the evolution of the state, but the scope of this influence is limited to within an individual episode.

Immediately we should note that we have already studied a finite horizon MDP under different terminology in Example 2: the online shortest path problem. To see the connection simply view each choice of edge as sequential timesteps within the period (or episode); the state is the current vertex, the action is the next choice of edge and the horizon is the maximal number of decision stages. With this connection in mind we can express the problem of maximizing the cumulative rewards in a finite horizon MDP $\sum_{k=1}^{K} \sum_{h=0}^{H-1} r(s_{kh}, a_{kh})$ equivalently as a bandit problem over periods $t = 1, 2, .., K$ where each bandit period is an entire episode of MDP

interaction and each bandit action is a *policy* $\mu_t$ for use within that episode. By contrast, a naive application of Thompson sampling to reinforcement learning that resamples policies every timestep within an episode could be extremely inefficient as it does not perform deep exploration.

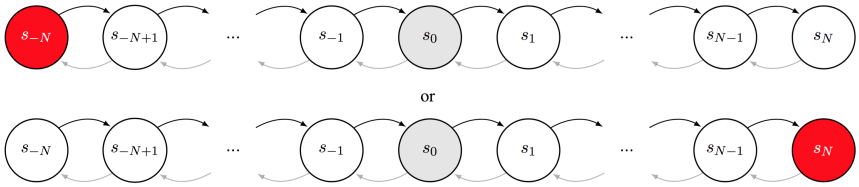


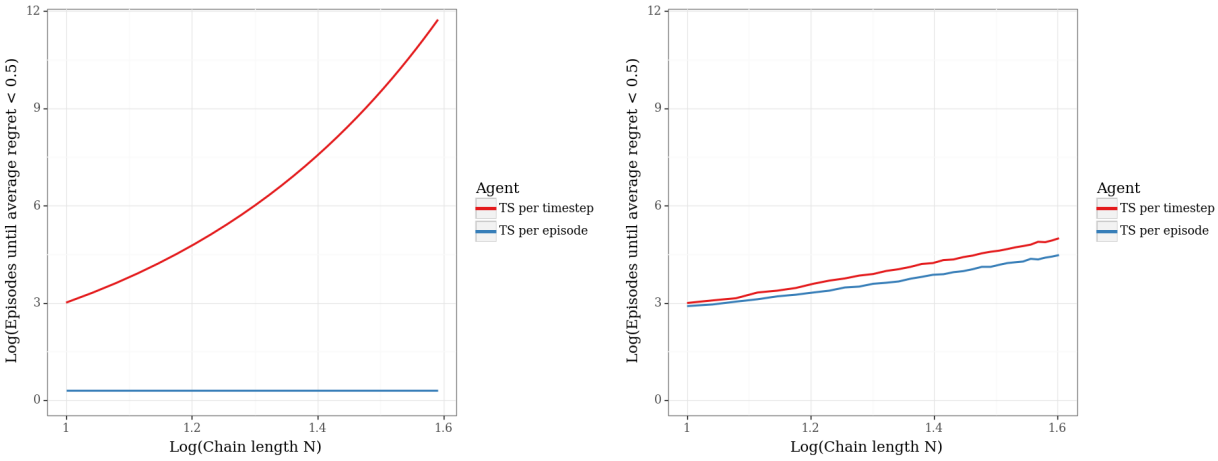

Figure 18: MDPs where Thompson sampling every timestep leads to inefficient exploration.

Consider the example in Figure 18 where the underlying MDP is characterized by a long chain of states $\{s_{-N}, .., s_N\}$ and only the one of the far left or far right positions are rewarding with equal probability; all other states produce zero reward and with known dynamics. Learning about the true dynamics of the MDP requires a consistent policy over $N$ steps right or $N$ steps left; a variant of Thompson sampling that resampled every timestep would be exponentially unlikely to make it to either end within $N$ steps [34]. By contrast, sampling at an episode level and holding that policy fixed for the duration of the episode would demonstrate deep exploration and so be able to learn the optimal policy within a single episode.

In order to apply Thompson sampling to policy selection we need a way of sampling from the posterior distribution for the optimal policy. One efficient way to do this, at least for finite $|\mathcal{S}|, |\mathcal{A}|$ is to maintain a posterior distribution over the reward distribution $R^M$ and the transition dynamics $P^M$ at each state and action $(s, a)$. In order to generate a sample for the optimal policy, simply take a single posterior sample for the reward distribution and transition probabilities and then solve for the optimal policy for this *sample*. This process is equivalent to maintaining a posterior distribution over the optimal policy, but may be more tractable depending on the problem setting. Estimating a posterior distribution over rewards is no different from the setting of bandit learning that we have already discussed at length within this paper. The transition function looks a little different, but for transitions over a finite state space the Dirichlet distribution is a useful conjugate prior. It is a multi-dimensional generalization of the Beta distribution from Example 3. The Dirichlet prior over outcomes in $\mathcal{S} = \{1, .., S\}$ is specified by a positive vector of pseudo-observations $\alpha \in \mathbb{R}^S_+$; updates to the Dirichlet posterior can be performed analytically simply by incrementing the appropriate column of $\alpha$ [4].

In Figure 19 we present a computational comparison of Thompson sampling where a new policy is drawn every 'timestep' and Thompson sampling where a new policy is drawn every 'episode'. This figure compares performance in the example shown in Figure 18. Figure 19a compares the performance of sampling schemes where the agent has an informative prior that matches the true underlying system. As explained above, sampling once per episode Thompson sampling is guaranteed to learn the true MDP structure in a single episode. By contrast, resampling every timestep leads to uniformly random actions until either $s_{-N}$ or $s_N$ is visited. Therefore, it takes a minimum of $2^N$ epsiodes for the first expected reward.

The difference in performance demonstrated by Figure 19a is particularly extreme because the prior structure means that there is only value to deep exploration, and none to 'shallow' exploration per timestep [34]. In Figure 19b we present results for Thompson sampling variant on the same environment but with uniform Dirichlet prior over transitions and $N(0, 1)$ prior over rewards in each state and action. With this prior structure Thompson sampling per timestep is not as hopeless, but still performs worse than Thompson sampling per episode. Once again, this difference increases with MDP problem size. Overall, Figure 19 demonstrates that the benefit of sampling per episode, rather than per timestep, can become arbitrarily large. As an additional benefit this approach is also more computationally efficient, since we only need to solve for the




(a) Using informed prior.

(b) Using uninformed prior.

Figure 19: Comparing Thompson sampling by episode with Thompson sampling by episode.

optimal policy once every episode rather than at each timestep.

# 8 Why it Works, When it Fails, and Alternative Approaches

Earlier sections demonstrate that Thompson sampling approaches can be adapted to address a number of problem classes of practical import. In this section, we provide intuition for why Thompson sampling explores efficiently, and briefly review theoretical work that formalizes this intuition. We will then highlight problem classes for which Thompson sampling is poorly suited, and refer to some alternative algorithms.

## 8.1 Why Thompson Sampling Works

To understand whether Thompson sampling is well suited to a particular application, it is useful to develop a high level understanding of why it works. As information is gathered, beliefs about arms' payouts are carefully tracked. By sampling actions according to the posterior probability they are optimal, the algorithm continues to sample all arms that could plausibly be optimal, while shifting sampling away from those that are extremely unlikely to be optimal. Roughly speaking, the algorithm tries all promising actions while gradually discarding those that are believed to under-perform.

This intuition is formalized in recent theoretical analyses of Thompson sampling. [5] observed empirically that for the simple Bernoulli bandit problem described in Example 1, the regret of Thompson sampling scales in an asymptotically optimal manner in the sense defined by [36]. A series of papers provided proofs confirming this finding [37, 38, 39]. Later papers have studied Thompson sampling as a general tool for exploration in more complicated online optimization problems. The first regret bounds for such problems were established by [40] for linear contextual bandits and by [41, 42] for an extremely general class of bandit models. Subsequent papers have further developed this theory [43, 44, 45] and have studied extensions of Thompson sampling to reinforcement-learning [35, 46, 47, 48].

## 8.2 Limitations of Thompson Sampling

Thompson sampling is a simple and effective method for exploration in broad classes of problems, but no heuristic works well for all problems. For example, Thompson sampling is certainly a poor fit for sequential learning problems that do not require much active exploration; in such cases by greedier algorithms that don't invest in costly exploration usually provide better performance. We now highlight two more subtle problem features that pose challenges when applying standard Thompson sampling.

### 8.2.1 Time Preference

Thompson sampling is effective at minimizing the exploration costs required to converge on an optimal action. It may perform poorly, however, in time-sensitive learning problems where it is better to exploit a high performing suboptimal action than to invest resources exploring arms that might offer slightly improved performance.

To understand this issue, let us revisit the motivating story given at the beginning of the paper. Suppose that while waiting for his friends to finish a game in another part of the casino, a gambler sits down at a slot machine with $k$ arms yielding uncertain payouts. For concreteness, assume as in Example 3 that the machine's payouts are binary with Beta distributed priors. But now, suppose the number of plays $\tau$ the gambler will complete before his friends arrive is uncertain, with $\tau \sim$ Geometric$(1-\delta)$ and $\mathbb{E}[\tau] = 1/(1-\delta)$. Is Thompson sampling still an effective strategy for maximizing the cumulative reward earned? This depends on the values of $\delta$ and $k$. When $\delta \to 1$ so $\mathbb{E}[\tau] \gg k$, it is generally worth exploring to identify the optimal arm so it can be played in subsequent periods. However, if $\mathbb{E}[\tau] < k$, there is not time to play every arm, and the gambler would be better off exploiting the best arm among those he tries initially than continuing to explore alternatives.

Related issues also arise in the nonstationary learning problems described in Section 6.3. When a nonstarionary system evolves rapidly, information gathered quickly becomes irrelevant to optimizing future performance. In such cases, it may be impossible to converge on the current optimal action before the system changes substantially, and the algorithms presented in Section 6.3 might perform better if they are modified to explore less aggressively.

This issue is discussed further in [49]. That paper proposes and analyzes *satisficing Thompson sampling*, a variant of Thompson sampling that is designed to minimize the exploration costs required to identify an action that is sufficiently close to optimal.

### 8.2.2 Problems Requiring Careful Assessment of Information Gain

Thompson sampling is well suited to problems where the best way to learn which action is optimal is to test the most promising actions. However, there are natural problems where such a strategy is far from optimal, and efficient learning requires a more careful assessment of the information actions provide. The following example is designed to make this point transparent.

> **Example 9** (A revealing action) Suppose there are $k+1$ actions $\{0, 1, ..., k\}$, and $\theta$ is an unknown parameter drawn uniformly at random from $\Theta = \{1, .., k\}$. Rewards are deterministic conditioned on $\theta$, and when played action $i \in \{1, ..., k\}$ always yields reward 1 if $\theta = i$ and 0 otherwise. Action 0 is a special "revealing" action that yields reward $1/2\theta$ when played.

Note that action 0 is known to never yield the maximal reward, and is therefore never selected by Thompson sampling. Instead, TS will select among actions $\{1, ..., k\}$, ruling out only a single action at a time until a reward 1 is earned and the optimal action is identified. An optimal algorithm for this problem would recognize that although action 0 cannot yield the maximal reward, sampling it is valuable because of the information it provides about other actions.

Indeed, by sampling action 0 in the first period, the decision maker immediately learns the value of $\theta$, and can exploit that knowledge to play the optimal action in all subsequent periods.

This example is described in [50], which also presents two problems of greater practical significance that require careful assessment of information: one related to recommendation systems and one involving bandits with sparse-linear reward models. That paper also proposes an algorithm that carefully assesses the information actions reveal.

### 8.3 Alternative Approaches

Much of the the work on multi-armed bandit problems has focused on problems with a finite number of independent arms, like Example 3 in this paper. For such problems, the Gittins index theorem [51] characterizes a Bayes optimal policy, which exactly maximizes expected cumulative discounted reward. Computing this policy requires solving an optimal stopping problem for each period and arm, and is much more demanding than Thompson sampling. For more complicated problems, the Gittins index theorem fails to hold, and computing an optimal policy is typically infeasible. A thorough treatment of Gittins indices is provided in [52].

Upper-confidence bound (UCB) algorithms offer another approach to efficient exploration. As the name suggests, these algorithms maintain upper-confidence bounds, representing the largest mean-reward an arm could plausibly generate given past observations. The algorithm then selects the action with the highest upper-confidence bound. At a high level, these algorithms are similar to Thompson sampling, in that they continue sampling all promising arms while gradually discarding those that under-perform. A more formal link between the two approaches is established in [42]. UCB algorithms have been proposed for a variety of problems, including bandit problems with independent arms [36, 53, 54, 55], bandit problems with linearly parameterized arms [30, 56], bandits with continuous action spaces and smooth reward functions [57, 58, 59], and exploration in reinforcement learning [60].

The knowledge gradient [61, 62] and information-directed sampling [50] are alternative algorithms that attempt to reason more carefully about the value of information acquired by sampling an action. Finally, there is a large literature on multi-armed bandit optimization in adversarial environments, which we will not review here. See [63] for thorough coverage.

### Acknowledgements

This work was generously supported by a research grant from Boeing, a Marketing Research Award from Adobe, and a Stanford Graduate Fellowship. We thank Susan Murphy for helpful suggestions. Section 7.3 draws material from a paper coauthored by Xiuyuan Lu. We thank her for help with the experiments presented in that section and for the associated code.

## References

[1] W.R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[2] William R Thompson. On the theory of apportionment. *American Journal of Mathematics*, 57(2):450–456, 1935.

[3] Jeremy Wyatt. *Exploration and inference in learning from reinforcement*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics., 1997.

[4] Malcolm Strens. A Bayesian framework for reinforcement learning. In *ICML*, pages 943–950, 2000.

[5] O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. In *Neural Information Processing Systems (NIPS)*, 2011.

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

[7] Kris Johnson Ferreira, David Simchi-Levi, and He Wang. Online network revenue management using Thompson sampling. *Working Paper*, 2016.

[8] Eric M Schwartz, Eric T Bradlow, and Peter S Fader. Customer acquisition via display advertising using multi-armed bandit experiments. *Marketing Science*, 2017.

[9] Aijun Bai, Feng Wu, and Xiaoping Chen. Bayesian mixture modelling and inference based Thompson sampling in Monte-Carlo tree search. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2013.

[10] T. Graepel, J.Q. Candela, T. Borchert, and R. Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 13–20, 2010.

[11] Deepak Agarwal. Computational advertising: the linkedin way. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1585–1586. ACM, 2013.

[12] Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. Laser: A scalable response prediction platform for online advertising. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 173–182. ACM, 2014.

[13] Jaya Kawale, Hung H Bui, Branislav Kveton, Long Tran-Thanh, and Sanjay Chawla. Efficient Thompson sampling for online matrix-factorization recommendation. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2015.

[14] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Asynchronous parallel Bayesian optimisation via Thompson sampling. *arXiv preprint arXiv:1705.09236*, 2017.

[15] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[16] Steven L Scott. Multi-armed bandit experiments in the online service economy. *Applied Stochastic Models in Business and Industry*, 31(1):37–45, 2015.

[17] Benjamin Van Roy Abbas Kazerouni Zheng Wen Ian Osband, Dan Russo. TS Tutorial: A tutorial on thompson sampling, 2017.

[18] George Casella and Edward I George. Explaining the Gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[19] Gareth O Roberts and Richard L Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, pages 341–363, 1996.

[20] Gareth O Roberts and Jeffrey S Rosenthal. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, 1998.

[21] Sébastien Bubeck, Ronen Eldan, and Joseph Lehec. Sampling from a log-concave distribution with projected Langevin Monte Carlo. *arXiv preprint arXiv:1507.02564*, 2015.

[22] Alain Durmus and Eric Moulines. Sampling from strongly log-concave distributions with the unadjusted Langevin algorithm. *arXiv preprint arXiv:1605.01559*, 2016.

[23] Xiang Cheng and Peter Bartlett. Convergence of Langevin MCMC in KL-divergence. *arXiv preprint arXiv:1705.09048*, 2017.

[24] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, 2011.

[25] Yee Whye Teh, Alexandre H Thiery, and Sebastian J Vollmer. Consistency and fluctuations for stochastic gradient Langevin dynamics. *Journal of Machine Learning Research*, 17(7):1–33, 2016.

[26] Carlos Gómez-Uribe. Online algorithms for parameter mean and variance estimation in dynamic regression. *arXiv preprint arXiv:1605.05697v1*, 2016.

[27] Xiuyuan Lu and Benjamin Van Roy. Ensemble sampling. *arXiv preprint arXiv:1705.07347*, 2017.

[28] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 87–94. ACM, 2008.

[29] Branislav Kveton, Csaba Szepesvari, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 767–776, 2015.

[30] V. Dani, T.P. Hayes, and S.M. Kakade. Stochastic linear optimization under bandit feedback. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT)*, pages 355–366, 2008.

[31] Ian Osband and Benjamin Van Roy. On optimistic versus randomized exploration in reinforcement learning. In *Proceedings of The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*. 2017.

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[33] Michael L Littman. Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553):445–451, 2015.

[34] Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[35] I. Osband, D. Russo, and B. Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013.

[36] T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[37] S. Agrawal and N. Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT)*, 2012.

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

[39] E. Kauffmann, N. Korda, and R. Munos. Thompson sampling: an asymptotically optimal finite time analysis. In *International Conference on Algorithmic Learning Theory*, 2012.

[40] S. Agrawal and N. Goyal. Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of The 30th International Conference on Machine Learning*, pages 127–135, 2013.

[41] D. Russo and B. Van Roy. Eluder dimension and the sample complexity of optimistic exploration. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2256–2264. Curran Associates, Inc., 2013.

[42] D. Russo and B. Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

[43] A. Gopalan, S. Mannor, and Y. Mansour. Thompson sampling for complex online problems. In *Proceedings of The 31st International Conference on Machine Learning*, pages 100–108, 2014.

[44] D. Russo and B. Van Roy. An information-theoretic analysis of Thompson sampling. *Journal of Machine Learning Research*, 17(68):1–30, 2016.

[45] Marc Abeille and Alessandro Lazaric. Linear Thompson sampling revisited. In *AISTATS 2017-20th International Conference on Artificial Intelligence and Statistics*, 2017.

[46] Aditya Gopalan and Shie Mannor. Thompson sampling for learning parameterized Markov decision processes. In *COLT*, pages 861–898, 2015.

[47] Ian Osband, Benjamin Van Roy, and Zheng Wen. Generalization and exploration via randomized value functions. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2377–2386, 2016.

[48] Michael Jong Kim. Thompson sampling for stochastic control: The finite parameter case. *IEEE Transactions on Automatic Control*, 2017.

[49] Daniel Russo, David Tse, and Benjamin Van Roy. Time-sensitive bandit learning and satisficing Thompson sampling. *arXiv preprint arXiv:1704.09028*, 2017.

[50] D. Russo and B. Van Roy. Learning to optimize via information-directed sampling. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1583–1591. Curran Associates, Inc., 2014.

[51] J.C. Gittins and D.M. Jones. A dynamic allocation index for the discounted multiarmed bandit problem. *Biometrika*, 66(3):561–565, 1979.

[52] J. Gittins, K. Glazebrook, and R. Weber. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons, Ltd, 2011.

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

[54] O. Cappé, A. Garivier, O.-A. Maillard, R. Munos, and G. Stoltz. Kullback-Leibler upper confidence bounds for optimal sequential allocation. *Annals of Statistics*, 41(3):1516–1541, 2013.

[55] E. Kaufmann, O. Cappé, and A. Garivier. On Bayesian upper confidence bounds for bandit problems. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[56] P. Rusmevichientong and J.N. Tsitsiklis. Linearly parameterized bandits. *Mathematics of Operations Research*, 35(2):395–411, 2010.

[57] R. Kleinberg, A. Slivkins, and E. Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, 2008.

[58] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, June 2011.

[59] N. Srinivas, A. Krause, S.M. Kakade, and M. Seeger. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250 –3265, may 2012.

[60] T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

[61] P.I. Frazier, W.B. Powell, and S. Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

[62] P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

[63] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and trends in machine learning*, 5(1):1–122, 2012.