3

Thompson Sampling for the Bernoulli Bandit

To digest how Thompson sampling (TS) works, it is helpful to begin with a simple context that builds on the Bernoulli bandit of Example 1.1 and incorporates a Bayesian model to represent uncertainty.

Example 3.1. (Beta-Bernoulli Bandit) Recall the Bernoulli bandit of Example 1.1. There are K actions. When played, an action k produces a reward of one with probability θ_k and a reward of zero with probability $1 - \theta_k$. Each θ_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 θ_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 θ_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 Γ 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 action are updated. The parameters (α_k, β_k) are sometimes called pseudo-counts, since α_k or β_k increases by one with each observed success or failure, respectively. A beta distribution with parameters (α_k, β_k) has mean $\alpha_k/(\alpha_k + \beta_k)$, and the distribution becomes more concentrated as $\alpha_k + \beta_k$ grows. Figure 2.2 plots probability density functions of beta distributions with parameters $(\alpha_1, \beta_1) = (601, 401)$, $(\alpha_2, \beta_2) = (401, 601)$, and $(\alpha_3, \beta_3) = (2, 3)$.

Algorithm 1 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 θ_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 α_{x_t} and β_{x_t} are updated.

TS, 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 α_k and β_k, rather than taken to be the expectation $\alpha_k/(\alpha_k+\beta_k)$. To avoid a common misconception, it is worth emphasizing TS 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.

Algorithm 1 BernGreedy(K, α, β)

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

Algorithm 2 BernTS(K, α, β)

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

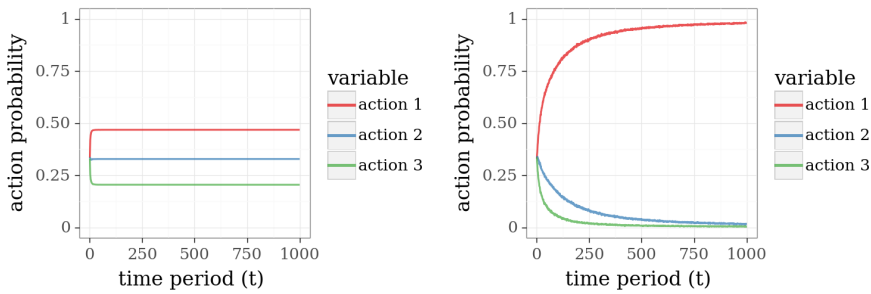
  #select and apply action:
  $x_t \leftarrow \text{argmax}_k \hat{\theta}_k$
  Apply x_t and observe r_t

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

To understand how TS improves on greedy actions with or without dithering, recall the three armed Bernoulli bandit with posterior distributions illustrated in Figure 2.2. 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. TS, 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, TS 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 TS 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 3.1 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

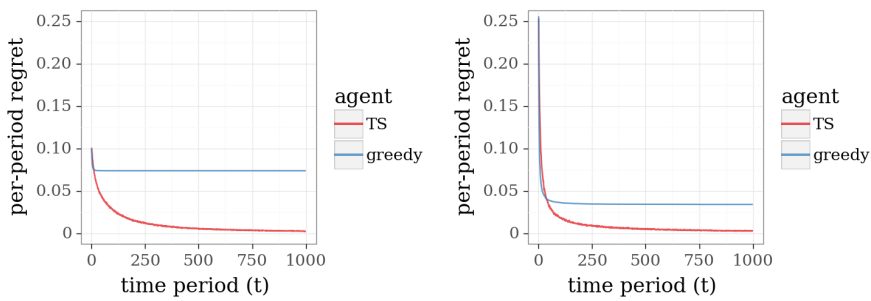
Figure 3.1: Probability that the greedy algorithm and Thompson sampling selects an action.

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. TS, 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, TS 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 3.2a plots per-period regret realized by the greedy algorithm and TS, again averaged over ten thousand simulations. The average

per-period regret of TS 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 θ. As such, it is often useful to also examine regret averaged over plausible values of θ. A natural approach to this involves sampling many instances of θ from the prior distributions and generating an independent simulation for each. Figure 3.2b 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 3.2a, though regret in Figure 3.2a is generally smaller over early time periods and larger over later time periods, relative to Figure 3.2b. 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 θ, and therefore the regret of randomly selected actions is smaller. The fact that per-period regret of TS is larger in Figure 3.2a than Figure 3.2b over later time periods, like period 1000, 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 θ.



(a) $\theta = (0.9, 0.8, 0.7)$ **(b)** average over random θ

Figure 3.2: Regret from applying greedy and Thompson sampling algorithms to the three-armed Bernoulli bandit.

4

General Thompson Sampling

TS 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 θ and represents his uncertainty using a prior distribution p.

Algorithms 3 and 4 present greedy and TS 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 θ with respect to the distribution p, while TS 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 θ 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 α and β, 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 Γ denotes the gamma function. In other words, under the prior distribution, components of θ are independent and beta-distributed, with parameters α and β.

For this problem, the greedy algorithm (Algorithm 3) and TS (Algorithm 4) begin each tth iteration with posterior parameters (α_k, β_k) for $k \in \{1, \ldots, K\}$. The greedy algorithm sets $\hat{\theta}_k$ to the expected value $\mathbb{E}_p[\theta_k] = \alpha_k/(\alpha_k + \beta_k)$, whereas TS randomly draws $\hat{\theta}_k$ from a beta distribution with parameters (α_k, β_k). 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 1.2.

Example 4.1. (Independent Travel Times) Recall the shortest path problem of Example 1.2. 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 θ_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 θ_e is independent and log-Gaussian-distributed with parameters μ_e and σ_e^2. That is, $\ln(\theta_e) \sim N(\mu_e, \sigma_e^2)$ is Gaussian-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 log-Gaussian-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 θ_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 log-Gaussian prior, this can be accomplished by setting $\mu_e = \ln(d_e) - \sigma_e^2/2$. Note that the

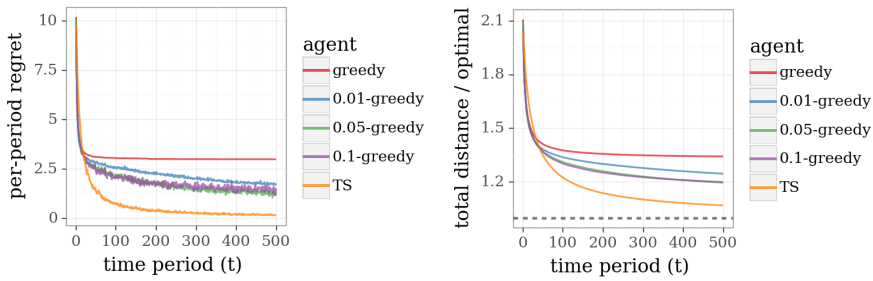
parameters μ_e and σ_e^2 also express a degree of uncertainty; in particular, the prior variance of mean travel time along an edge is $(e^{\sigma_e^2} - 1)d_e^2$.

The greedy algorithm (Algorithm 3) and TS (Algorithm 4) can be applied to Example 4.1 in a computationally efficient manner. Each algorithm begins each tth iteration with posterior parameters (μ_e, σ_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 TS randomly draws $\hat{\theta}_e$ from a log-Gaussian distribution with parameters μ_e and σ_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 (μ_e, σ_e^2), for each $e \in E$, are updated according to (4.3).

Figure 4.1 presents results from applying greedy and TS algorithms to Example 4.1, with the graph taking the form of a binomial bridge, as shown in Figure 4.2, 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 TS converges quickly to optimal, while that is far from true for the greedy algorithm. We also plot results generated by ϵ-greedy exploration, varying ϵ. 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 TS. This is because ϵ-greedy exploration is not judicious in how it selects paths to explore. TS, 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 θ. With TS, this converges to one at a respectable rate. The same can not be said for ϵ-greedy approaches.



(a) regret **(b)** cumulative travel time vs. optimal

Figure 4.1: Performance of Thompson sampling and ϵ-greedy algorithms in the shortest path problem.

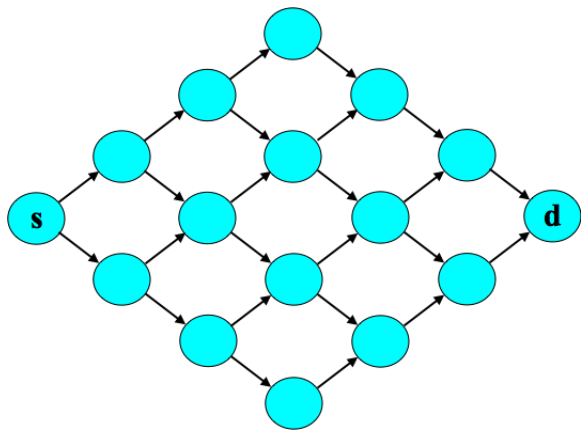


Figure 4.2: A binomial bridge with six stages.

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 more complex variation of the binomial bridge example.

Example 4.2. (Correlated Travel Times) As with Example 4.1, let each θ_e be independent and log-Gaussian-distributed with parameters μ_e and σ_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, η_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 $\zeta_{t,e}$, η_t, $\nu_{t,0}$, and $\nu_{t,1}$ to be independent log-Gaussian-distributed with parameters $-\tilde{\sigma}^2/6$ and $\tilde{\sigma}^2/3$. The distributions of the shocks $\zeta_{t,e}$, η_t, $\nu_{t,0}$ and $\nu_{t,1}$ are known, and only the parameters θ_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 4.1, though the joint distribution over $y_t|\theta$ differs.

The common factors induce correlations among travel times in the binomial bridge: η_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

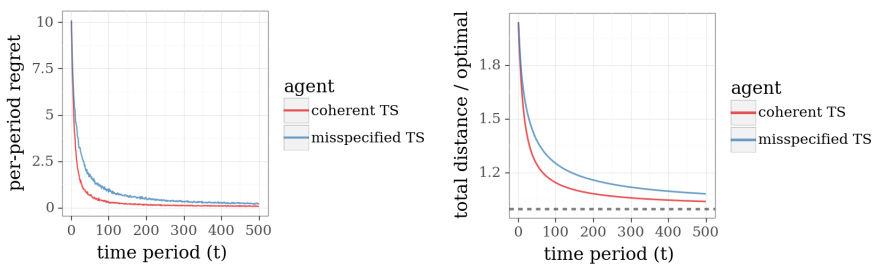
$$\tilde{C}_{e,e'} = \begin{cases} \tilde{\Sigma}^{-1}_{e,e'} & \text{if } e, e' \in x_t \\ 0 & \text{otherwise,} \end{cases}$$

for $e, e' \in E$. Then, the posterior distribution of ϕ is Gaussian with a mean vector μ and covariance matrix Σ that can be updated according

to

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

TS (Algorithm 4) can again be applied in a computationally efficient manner. Each tth 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 Gaussian distribution with mean μ and covariance matrix Σ, 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 (μ, Σ) are updated according to (4.4).



(a) regret **(b)** cumulative travel time vs. optimal

Figure 4.3: Performance of two versions of Thompson sampling in the shortest path problem with correlated travel times.

Figure 4.3 plots results from applying TS to Example 4.2, 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 4.1. In particular, the agent could maintain an independent log-Gaussian posterior for each θ_e, updating parameters (μ_e, σ_e^2) as though each $y_{t,e}|\theta$ is independently drawn from a log-Gaussian distribution. As a baseline for comparison, Figure 4.3 additionally plots results from application of this approach, which we will refer to here as *misspecified TS*. 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 TS*. 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.

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 5.1. (Binary Feedback) Consider Example 4.2, except with deterministic travel times and noisy binary observations. Let the graph represent a binomial bridge with M stages. Let each θ_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 5.1 plots results from application of three approximate versions of TS to an online shortest path problem on a twenty-stage binomial bridge with binary feedback. The algorithms leverage Langevin Monte Carlo, the Laplace approximation, and the bootstrap, three approaches we will discuss, and the results demonstrate effective learning, in the sense that regret vanishes over time. Also plotted as a baseline for comparison are results from application of the greedy algorithm.

In the remainder of this section, we will describe several approaches to approximate TS. 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 θ conditioned on the history $\mathbb{H}_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$ of observations. TS 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 TS when exact posterior sampling is infeasible.

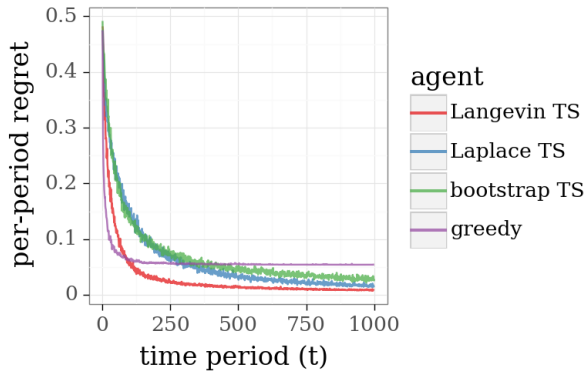


Figure 5.1: Regret experienced by approximation methods applied to the path recommendation problem with binary feedback.

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 $(\hat{\theta}^n : n = 0, 1, 2, \ldots)$ 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 nth 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 kth 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 (Casella and George, 1992) 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 sample 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 Laplace Approximation

We now discuss an approach that 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. Chapelle and Li (Chapelle and Li, 2011) proposed this method to approximate TS 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 Gaussian-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 5.1. Bayes rule implies that the posterior density f_{t-1} of θ 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 Gaussian distribution with mean $\overline{\theta}$ and covariance matrix $(-\nabla^2 \ln(f_{t-1}(\overline{\theta})))^{-1}$.

Laplace approximations are well suited for Example 5.1 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 4.2. For this problem, posterior distributions components of θ are log-Gaussian. However, the distribution of ϕ, where $\phi_e = \ln(\theta_e)$ for each edge $e \in E$, is Gaussian. As such, if the Laplace approximation approach is applied to generate a sample $\hat{\phi}$ from the posterior distribution of ϕ, the Gaussian 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 θ. 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.

To produce the computational results reported in Figure 5.1, we applied Newton's method with a backtracking line search to maximize

$\ln(f_{t-1})$. Though regret decays and should eventually vanish, it is easy to see from the figure that, for our example, the performance of the Laplace approximation falls short of Langevin Monte Carlo, which we will discuss in the next section. This is likely due to the fact that the posterior distribution is not sufficiently close to Gaussian. It is interesting that, despite serving as a popular approach in practical applications of TS (Chapelle and Li, 2011; Gómez-Uribe, 2016), the Laplace approximation can leave substantial value on the table.

5.3 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 dynamics 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 as its unique stationary distribution, and under reasonable technical conditions, the distribution of ϕ_t converges rapidly to this stationary distribution (Roberts and Tweedie, 1996; Mattingly *et al.*, 2002). Therefore simulating the process (5.1) provides a means of approximately sampling from g.

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 $W_1, W_2, \ldots$ are i.i.d. standard Gaussian random variables and $\epsilon > 0$ is a small step size. Like a gradient ascent method, under this method ϕ_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 ϕ_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 (Roberts and Rosenthal, 1998; Bubeck *et al.*, 2018; Durmus and Moulines, 2016; Cheng and Bartlett, 2018). These papers typically require ϵ is sufficiently small, or that a decaying sequence of step sizes $(\epsilon_1, \epsilon_2, \ldots)$ is used.

We make two standard modifications to this method to improve computational efficiency. First, following recent work (Welling and Teh, 2011), we implement *stochastic gradient* Langevin Monte Carlo, which uses sampled minibatches of data to compute approximate rather than exact gradients. Our implementation uses a mini-batch size of 100; this choice seems to be effective but has not been carefully optimized. When fewer than 100 observations are available, we follow the Markov chain (5.2) with exact gradient computation. When more than 100 observations have been gathered, we follow (5.2) but use an estimated gradient $\nabla \ln(\hat{g}_n(\phi_n))$ at each step based on a random subsample of 100 data points. Some work provides rigorous guarantees for stochastic gradient Langevin Monte Carlo by arguing the cumulative impact of the noise in gradient estimation is second order relative to the additive Gaussian noise (Teh *et al.*, 2016).

Our second modification involves the use of a preconditioning matrix to improve the mixing rate of the Markov chain (5.2). For the path recommendation problem in Example 5.1, 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 should leverage second order information. Similarly, due to poor conditioning, we may need to choose an extremely small step size ϵ, 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 A by simulating the Markov chain

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

where $A^{1/2}$ denotes the matrix square root of A. In our implementation, we take $\phi_0 = \text{argmax}_\phi \ln(g(\phi))$, so the chain is initialized at the posterior mode, computed via means discussed in Section 5.2, and take the preconditioning matrix $A = -(\nabla^2 \ln(g(\phi))|_{\phi=\phi_0})^{-1}$ to be the negative inverse Hessian at that point. It may be possible to improve

computational efficiency by constructing an incremental approximation to the Hessian, as we will discuss in Subsection 5.6, but we do not explore that improvement here.

5.4 Bootstrapping

As an alternative, we discuss an approach based on the statistical bootstrap, which accommodates even very complex densities. Use of the bootstrap for TS was first considered in (Eckles and Kaptein, 2014), though the version studied there applies to Bernoulli bandits and does not naturally generalize to more complex problems. There are many other 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 tutorial.

Like the Laplace approximation approach, our bootstrap method assumes that θ 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 θ. 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 θ 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 θ^0 from the prior distribution f_0. Let Σ 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 Gaussian with mean θ^0 and covariance matrix Σ, 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 θ_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 5.1, 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.

As can be seen from Figure 5.1, for our example, bootstrapping performs about as well as the Laplace approximation. One advantage of the bootstrap is that it is nonparametric, and may work reasonably regardless of the functional form of the posterior distribution, whereas the Laplace approximation relies on a Gaussian approximation and Langevin Monte Carlo relies on log-concavity and other regularity assumptions. That said, it is worth mentioning that there is a lack of theoretical justification for bootstrap approaches or even understanding of whether there are nontrivial problem classes for which they are guaranteed to perform well.

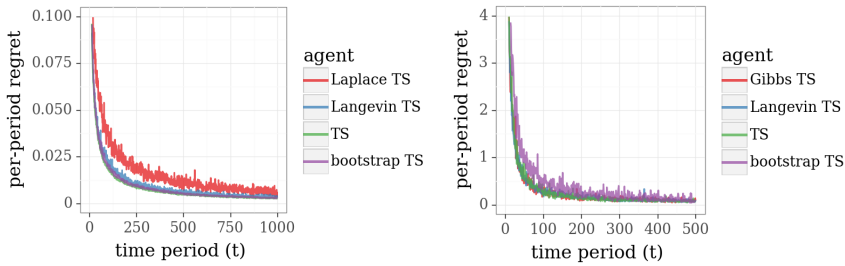
5.5 Sanity Checks

Figure 5.1 demonstrates that Laplace approximation, Langevin Monte Carlo, 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 TS would offer substantially better performance. Since we do not have a tractable means of carrying out exact TS for this problem, in this section, we apply our approximation methods to problems for which exact TS 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 TS algorithms were reported in Figure 3.2(b). For this problem, components of θ 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 TS. Figure 5.2a plots results from applying Laplace approximation, Langevin Monte Carlo, and bootstrap approaches. For this problem, our approximation methods offer performance that is qualitatively similar to exact TS, though the Laplace approximation performs marginally worse than alternatives in this setting.

Next, consider the online shortest path problem with correlated edge delays. Regret experienced by TS applied to such a problem were reported in Figure 4.3a. As discussed in Section 5.2, applying the Laplace approximation approach with an appropriate variable substitution leads to the same results as exact TS. Figure 5.2b compares those results to what is generated by Gibbs sampling, Langevin Monte Carlo, and bootstrap approaches. Again, the approximation methods yield competitive results, although bootstrapping is marginally less effective than others.

It is easy to verify that for the online shortest path problem and specific choices of step size $\epsilon = 1/2$ and conditioning matrix $A = \Sigma_t$, a single Langevin Monte Carlo iteration offers an exact posterior sample. However, our simulations do not use this step size and carry out multiple iterations. The point here is not to optimize results for our specific problem but rather to offer a sanity check for the approach.



(a) Bernoulli bandit **(b)** online shortest path

Figure 5.2: Regret of approximation methods versus exact Thompson sampling.

5.6 Incremental Implementation

For each of the three approximation methods we have discussed, the computation 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 TS 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 θ. 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 Gaussian and observations are linear in θ and perturbed only by Gaussian 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 is 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 (Gómez-Uribe, 2016) as a means for incremental approximate TS 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, TS 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 (Lu and Van Roy, 2017), which draws inspiration from work on exploration in deep reinforcement learning (Osband *et al.*, 2016a).

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_τ^n can loosely be interpreted as a number of replicas of the data sample (x_τ, y_τ) 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 7.4, we will discuss another, which we apply to active learning with neural networks.

6

Practical Modeling Considerations

Our narrative over previous sections has centered around a somewhat idealized view of TS, 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 TS 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 θ 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 1.1. There are K banner ads for a single product, with unknown click-through probabilities $(\theta_1, \ldots, \theta_K)$. Given a prior, TS can learn to select the most successful ad. We could use a uniform or, equivalently, a beta$(1, 1)$ distribution over each θ_k. However, if some values of θ_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 TS 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 kth partition consisting of those with stylistic features most similar to the kth ad under current consideration. Figure 6.1 plots a hypothetical empirical cumulative distribution of click-through probabilities for ads in the kth 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 6.1. Intuitively, this process assumes that click-through probabilities of past ads in set k represent plausible values of θ_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 6.2 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. TS 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

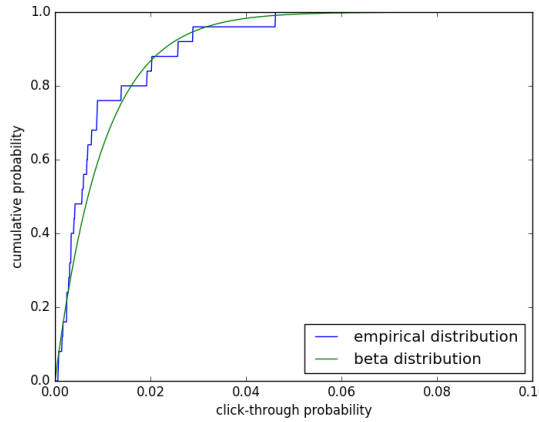
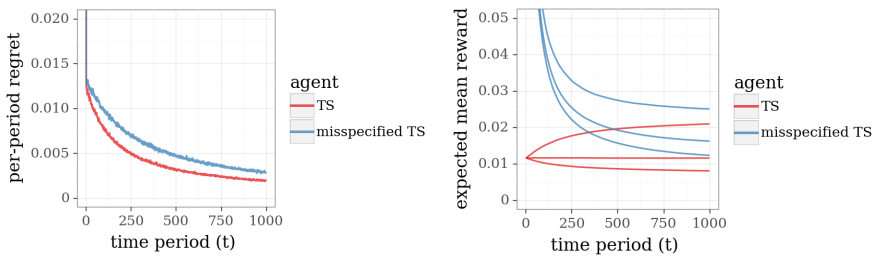


Figure 6.1: An empirical cumulative distribution and an approximating beta distribution.

mean rewards. Figure 6.2a plots expected regret, demonstrating that the misspecified prior increases regret. Figure 6.2a 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 6.2: Comparison of TS 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 θ except through possible dependence on the history of observations, TS (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 TS 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 tth 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 θ that are constant over time. As TS 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, TS remains an effective approach so long as model parameters change little over durations that are sufficient to identify effective actions.

In principle, TS could be applied to a broad range of problems where the parameters $\theta_1, \theta_2, \theta_3, \ldots$ evolve according to a stochastic process 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 τ 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 τ 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. Theory supporting such an approach is developed in (Besbes *et al.*, 2014).

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 θ_t. We now consider such a model and a suitable modification of TS. Let us start with

the simple context of a Bernoulli bandit. Take the prior for each kth mean reward to be beta(α, β). Let the algorithm update parameters to identify the belief distribution of θ_t conditioned on the history $\mathbb{H}_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$ according to

(6.1)

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

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 (α_k, β_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 γ 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 γ, $\overline{\alpha}$, and $\overline{\beta}$, and the formula used to update distribution parameters.

The more general form of TS presented in Algorithm 4 can be modified in an analogous manner. For concreteness, let us focus on the case where θ 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, γ controls the rate at which uncertainty is injected. The modified version of Algorithm 2, which we refer to as *nonstationary TS*, takes γ and $\overline{p}$ as additional arguments and replaces the distribution update formula.

Figure 6.3 illustrates potential benefits of nonstationary TS 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 TS updates its belief distribution in a manner consistent with the underlying system, TS pretends that the success probabilities are constant over time and updates its beliefs accordingly. As the system drifts over time, TS becomes less effective, while nonstationary TS retains reasonable performance. Note, however, that due to nonstationarity, no algorithm can promise regret that vanishes with time.

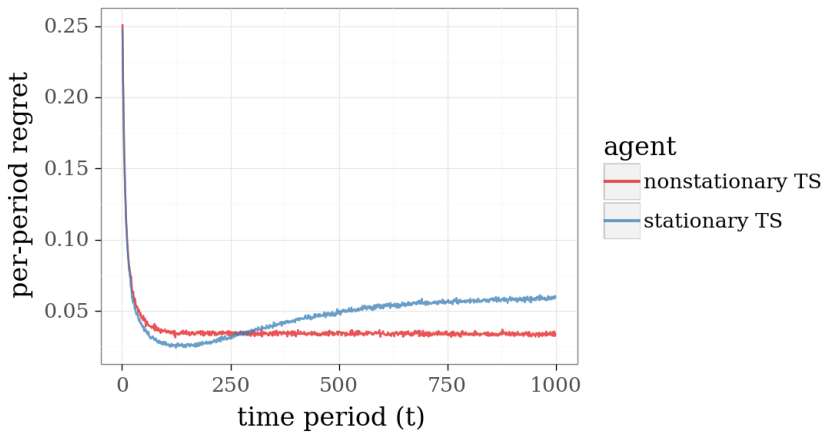


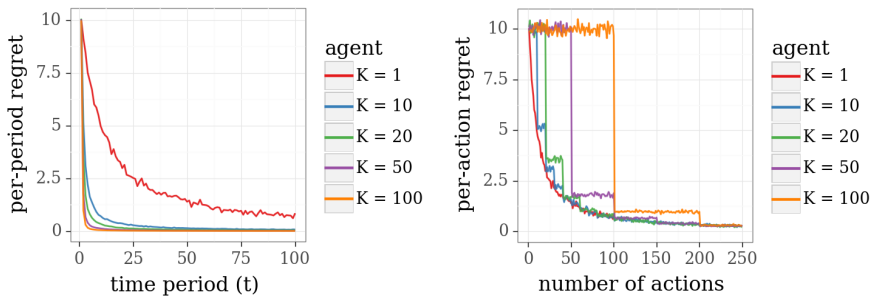
Figure 6.3: Comparison of TS versus nonstationary TS with a nonstationary Bernoulli bandit problem.

6.4 Concurrence

In many applications, actions are applied concurrently. As an example, consider a variation of the online shortest path problem of Example 4.1. In the original version of this problem, over each period, an agent selects and traverses a path from origin to destination, and upon completion, updates a posterior distribution based on observed edge traversal times.

Now consider a case in which, over each period, multiple agents travel between the same origin and destination, possibly along different paths, with the travel time experienced by agents along each edge e to conditionally independent, conditioned on θ_e. At the end of the period, agents update a common posterior distribution based on their collective experience. The paths represent concurrent actions, which should be selected in a manner that diversifies experience.

TS naturally suits this concurrent mode of operation. Given the posterior distribution available at the beginning of a time period, multiple independent samples can be drawn to produce paths for multiple agents. Figure 6.4 plots results from applying TS in this manner. Each simulation was carried out with K agents navigating over each time period through a twenty-stage binomial bridge. Figure 6.4(a) demonstrates that the per-action regret experienced by each agent decays more rapidly with time as the number of agents grows. This is due to the fact that each agent's learning is accelerated by shared observations. On the other hand, Figure 6.4(b) shows that per-action regret decays more slowly as a function of the number of actions taken so far by the collective of agents. This loss is due to fact that the the posterior distribution is updated only after K concurrent actions are completed, so actions are not informed by observations generated by concurrent ones as would be the case if the K actions were applied sequentially.



(a) per-action regret over time **(b)** per-action regret over actions

Figure 6.4: Performance of concurrent Thompson sampling.

As discussed in (Scott, 2010), concurrence plays an important role in web services, where at any time, a system may experiment by providing

different versions of a service to different users. Concurrent TS offers a natural approach for such contexts. The version discussed above involves synchronous action selection and posterior updating. In some applications, it is more appropriate to operate asynchronously, with actions selected on demand and the posterior distribution updated as data becomes available. The efficiency of synchronous and asynchronous variations of concurrent TS is studied in (Kandasamy *et al.*, 2018). There are also situations where an agent can alter an action based on recent experience of other agents, within a period before the action is complete. For example, in the online shortest path problem, an agent may decide to change course to avoid an edge if new observations made by other agents indicate a long expected travel time. Producing a version of TS that effectively adapts to such information while still exploring in a reliably efficient manner requires careful design, as explained in (Dimakopoulou and Van Roy, 2018).