Multi-Armed Bandit Problem

Sunith Suresh, Lin Xiao, Ilan Man and Sanjay Hariharan

May 4, 2016

1 Introduction

This paper explores the multi-armed bandit problem (MAB). A multi-armed bandit is a sequential experiment with the goal of achieving the largest possible reward from a payoff distribution with unknown parameters. The term multi-armed bandit comes from slot machines where each machine is known as a one-armed bandit. In this set up, at each iteration the player must decide which arm of the experiment to observe next.

The task is complicated by the stochastic nature of the bandits in the following two ways:

1. A suboptimal bandit can return many winnings, purely by chance, which would make us believe that it is a very profitable bandit. Similarly, the best bandit might not yield a reward if only played a few times.
2. If we have found a bandit that returns good results, do we keep drawing from it to maintain our good score, or do we try other bandits in hopes of finding an even better bandit? How do we know when to switch and when to stick to the current bandit? This is known as the *exploration vs. exploitation* dilemma.

This is a classic reinforcement learning problem in machine learning literature, as well as a game theoritic problem in economic theory.

This paper reviews several strategies for selecting machines, including a creative application of dynamic programming. Note that there are many variations on the stochastic MAB, including contextual bandit and adverserial bandit, which we will not discuss here and is outside of the scope of this paper.

2 Multi-armed Bandits

In machine learning literature, the stochastic multi-armed bandit problem is formulated as:

> Given K machines, each with an unknown probability of yielding a reward, which come from a fixed but unknown distribution parameterized by θ_i, for $i \in (1, ..., K)$, and N total plays, decide which machines to play in order to maximize the total reward.

2.1 Reget vs. Reward

It is common in the literature to express the maximum reward as minimzing *regret* compared to the best arm in hindsight, acting as a sort of opportunity cost. That is, define regret at each step as $r_t = R^*_t - R_{t,i}$, where R^* is the reward yielded by selecting the best machine and $R_{t,i}$ is the reward yielded by selecting machine i at time t. Note that in reality we don't know what the best machine is - this formulation is purely a way to compare the performance of algorithms with simulated data. The goal is then to devise a strategy to minimize $\sum_{t=1}^{N} r_t$. Note that since this is a stochastic problem, we aim to minimize regret in expectation or with high probability.

2.2 Exploitation vs. Exploration

The tension between playing all the machines many times and only playing the best machine is a key concern for optimizing an algorithm. The problem with exploring too much is that every time you play a machine that isn't the best, you incur some regret, since you won't be playing a machine with highest expected reward. In addition, if you view machines as, for example, websites that users click on, then every time you send users to a bad website, you risk losing them forever, if they have a very unfavorable experience. On the other hand, if you exploit too quickly, then you won't have explored the space of possible machines and may miss out on the most optimal machine. This could be very costly, especially if you don't change your mind later on - you will always incur some cost in the form of the regret function.

3 Bayesian approaches: Bernoulli Bandit

A popular approach to solving the MAB problem is to minimize regret using Bayesian inference. Specifically, assume that rewards are distributed as a Bernoulli with some latent parameter, θ_i, specific to each machine. In addition, we can put a prior distribution on θ_i and update our belief about it as we run the algorithm and learn which machines are better than others. Since rewards are generated from a Bernoulli distribution, a natural choice for the distribution on θ_i is a Beta. Formally stated:

$$\theta_i \sim \text{Beta}(\alpha_i, \beta_i)$$
$$r_i = \text{Reward from machine i} \sim \text{Bernoulli}(\theta_i)$$

Exploiting the conjugacy of the Beta-Bernoulli model, the posterior distribution of getting a reward from machine$_i$, after playing for N iterations is:

$$\text{Beta}(\alpha_i + R_N, \beta_i + T - R_N)$$

where $R_N = \sum_{t=1}^{N} r_t$.

Before beginning to play, we assume no prior knowledge about any machine's propensity to yield a positive reward. Therefore, we initially set $\alpha_i, \beta_i = 1$, which is a common objective prior for the Beta distribution.

3.1 Dynamic Programming

As mentioned above, our goal is to either minimize regret or maximize total expected rewards, $\mathbb{E}(R_N)$. Under the above setup, we constructed an approach to maximizing future expected rewards based on a backwards, dynamic programming scheme. Note that we derive the algorithm assuming $K = 2$ machines.

Before we discuss how to construct our acquisition function $U(\theta|X)$ which will help us decide which machine to play at each iteration, we note the following interesting observations:

1. At time $T = 1$, we have no prior belief concerning any machine. Thus, it will not matter which machine we pick the first time around, though it will influence all subsequent choices.
2. At time $T = N$, we are at our final trial, and thus are only concerned with the reward on this trial. As such, we opt for a **greedy** approach and pick the machine with the highest expected reward. Using a Beta posterior, that is: $\underset{i}{\text{argmax}}(\frac{\alpha_1}{\alpha_1+\beta_1}, \frac{\alpha_2}{\alpha_2+\beta_2})$
3. For times $T = 2, ..., N-1$, we must choose the machine based on the expected sum of future rewards, based on our framework above: $\mathbb{E}(\sum\limits_{i=1}^{N} r_i)$
4. In general, at time $T = t$, $\mathbb{E}(\sum\limits_{i=t}^{N} R_i | M_{1:t-1}, R_{1:t-1}) = \mathbb{E}(\sum\limits_{i=t+1}^{N} R_i|\theta) + \mathbb{E}(R_t|\theta)$
   - $M_{1:t}$ = machines chosen from times $1, ..., t$
   - $R_{1:t}$ = rewards given from times $1, ..., t$

Given we perform N trials, we construct our algorithm in a backwards fashion:

1. For $T = N$:

- For each node, we choose the greedy, i.e. the one with the maximum expected reward.

2. For $T = N - 1$:

   - Given that we know the most optimal choice for the last trial, we select the machine on this trial that will give us the maximum expected sum of future rewards
   - $\mathbb{E}(R_{N-1} + R_N)$, and we know R_N from the previous iteration

3. For $T = N - 2, ..., 2$:

   - We follow similar intuition as above, choosing the most optimal choice for the maximum expected sum of future rewards, given the optimal choice for all subseqent trials already calculated
   - $\mathbb{E}(\sum_{i=T}^{N} R_i) = \mathbb{E}(\sum_{i=t+1}^{N} R_i) + \mathbb{E}(R_t)$

4. For $T = 1$:

   - As mentioned above since the first iteration will not influence our decision, as we have no prior knowledge, we select any machine at random.

As an illustration, consider the following tree. Here we denote $f(\alpha_1, \beta_1, \alpha_2, \beta_2)$ as the posterior parameters of the machine at time T, starting from the beginning **CAN YOU ELABORATE ON WHAT f IS? WHAT'S THE OUTPUT? WHAT ARE THE BRANCHES EXACTLY?**:

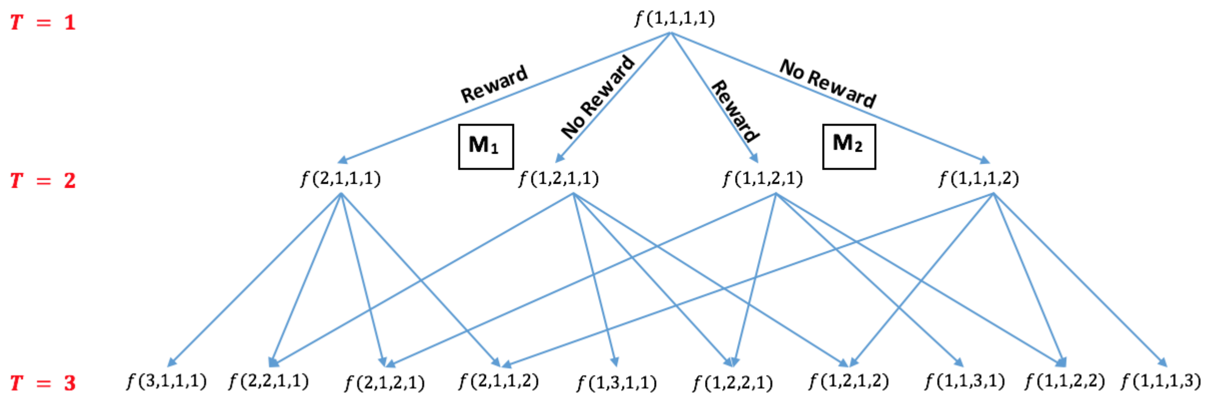


Figure 1: Forest of Possible Outcomes

As time increases, the complexity of possible branches increase exponentially, but it is important to note that some branches merge back together. Visually, our algorithm takes in two pieces of information:

1. Current parameters of the machine based on previous observations

2. Number of Trials we expect to perform

Our algorithm creates a tree of all possible future states, like above, and dynamically works backwards, assuming at the final stage we want to be completely greedy (since it is our last trial). The output of the algorithm is **the specific machine we should play at the current stage**, as well as the expected sum of rewards given you follow the algorithm at each trial.

One might assume that we would be greedy at every approach, but our algorithm proves otherwise. It bases its decision on expected future rewards, and it is interesting to see it choose the machine with the lower expected reward, based on its future possible states.**WHERE HAVE WE SEEN THIS? DO YOU HAVE AN EXAMPLE? ALSO WHY IS IT INTERESTING? I.E. BECAUSE WE WOULDN'T EXPECT IT BASED ON THE GREEDY APPROACH? MAYBE TALK ABOUT WHY WE DIDN'T IMPLEMENT THIS? AND CAN YOU ADD IN SOMETHING LEADING TO THE NEXT SECTIONS WHERE WE DISCUSS OTHER ALGORITHMS THAT WE IMPLEMENTED?**

3.2 Thompson Sampling

As mentioned in the algorithm above, our approach to solving the MAB problem is inherently Bayesian. Below we outline a popular approach used in practice called Thompson Sampling, which relies on Bayesian mechanics. The general algorithm is as follows:

```
1. For $t = 1, ..., N$
    (a) $U(\alpha_i, \beta_i) = i$ // select machine $i$ at time $t$ using the acquisition function
    (b) $r_t \sim \text{Bernoulli}(\theta_i)$ // We play machine $i$, and observe reward $r_t$
        • If $r_t = 1 : \alpha_i = \alpha_i + 1$ // increase our positive belief in machine $i$
        • else if $r_t = 0 : \beta_i = \beta_i + 1$ // increase our negative belief (i.e. failure) in machine $i$
    (c) $R_t = R_t + r_t$ // add reward at time $t$ to running total
```

This algorithm provides a framework for how to think about this problem in a Bayesian way. To maximize expected rewards, we must optimally choose the policy function U.

Below we review 3 ways to select the policy, U:

1. **Explore**: Randomly select one of the K machines, without considering how many win/losses we've seen. This is a pure exploratory strategy and is used for comparison purposes only. In reality no one would choose this approach.
2. **Exploit**: Sample each $\theta_i \sim Beta(\alpha_i, \beta_i)$. Select $\underset{i}{\text{argmax}}\ \mathbb{E}[\theta_i]$. This is called exploit because we are greedily choosing the best machine at each step based on expected rewards. Note however that even if we select the best expected machine, there is a probability of $\frac{\beta_i}{\alpha_i+\beta_i}$ that it fails to generate a reward. So the amount of exploration is much less than above but still non-zero. That this approach is known as *Adbandit*.
3. **Thompson-Sampling**: Sample each $\theta_i \sim Beta(\alpha_i, \beta_i)$. Select $\underset{i}{\text{argmax}}\ \theta_i$. This approach falls somewhere between Exploit and Explore, but is much closer to Exploit. In expectation we expect Thompson-Sampling to be very close to Exploit.

We ran these three approaches on a dataset of $K = 5$ machines with true reward distribution, $[0.05, 0.1, 0.3, 0.2, 0.5]$ and calculate the cumulative regret after 300 iterations. Below is the output for 1 particular run:

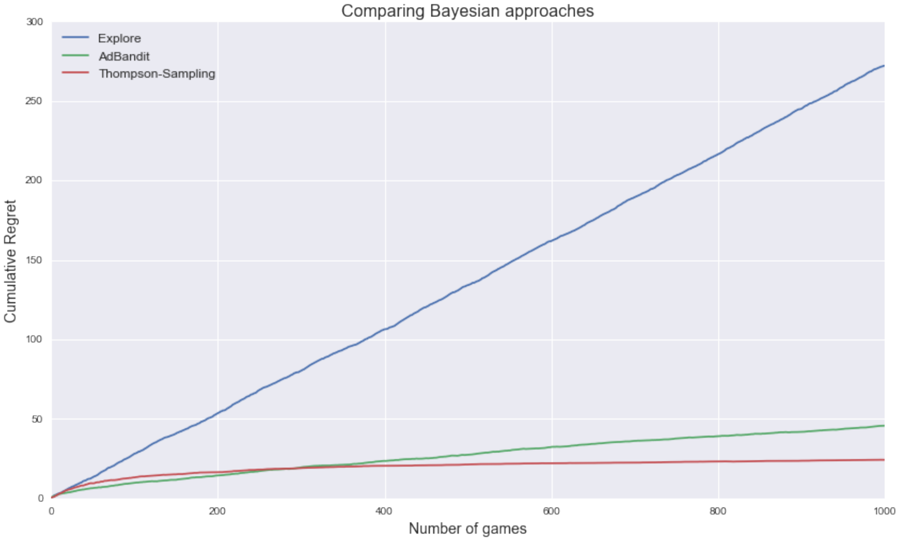


Figure 2: Cumulative Regret

As expected, the Explore strategy performs the worst. In this run, Adbandit performed the best. If we took the average of many runs, we'd expect Thompson-Sampling and Adbandit to perform similarly. Note that Adbandit finds the best machine very quickly and stays there. If K was larger, then it would be more exploring, or potentially get stuck in a poorly performing machine for a long time before finding a better one. We expect Thompson-Sampling to outperform Adbandit for larger K.

3.3 ϵ-greedy

The most simple approach to tackling the MAB problem is the ϵ-greedy algorithm. As explained earlier, the greedy algorithm always takes the best action at every trial. In other words, it simply chooses the bandit with highest expected reward at every iteration. The problem with this purely exploitative approach is that, it will take a large number of iteration to explore all the machines and converge onto the best machine. The ϵ-greedy algorithm is almost a greedy algorithm as it generally exploits the best machine, however every once in a while it explores other machines.

The parameter ϵ controls the percentage of iterations for which the algorithm explores machines. The algorithm can be described as follows: For a proportion $1 - \epsilon$ of the trials, choose the best machine(i.e the machine with highest expected reward), and for a proportion of ϵ explore other machines (with uniform probability).

Given machines $\{1, ..., K\}$ with initial empirical means $\{\hat{\mu_1}(0), ..., \hat{\mu_K}(0)\}$ For trials $t = \{1, ..., N\}$, the probability of choosing machine i is given by

$$p_i(t+1) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{K}, & \text{if } i = argmax_{j=1,..,K}\hat{\mu_j}(t), \\ \frac{\epsilon}{K}, & \text{otherwise.} \end{cases}$$

It follows from the algorithm that if $\epsilon = 0$, the algorithm reduces to a purely exploitative one, whereas if $\epsilon = 1$, it reduces to a purely explorative one. Therefore the value of ϵ, defined by the user, controls the level of exploration and exploitation employed by the algorithm.

The ϵ-greedy algorithm is considered to be suboptimal due to the constant value of ϵ. As the number of trials increase, asymptotically we can be reasonably certain as to which machine is the best machine from the observed rewards. However, the algorithm will always explore for ϵ percent of the trials. To address this issue, variants of the algorithm have been proposed. Two popular variants are known as ϵ-first algorithm and ϵ-decreasing algorithm.

The ϵ-first algorithm consists of doing the exploration all at once at the beginning and then switching to pure exploitation. Given N total trials, for ϵN trials, the machines are randomly chosen (with uniform probability). After the exploration phase, for the remaing $\big(1 - \epsilon\big)N$ trials, the algorithm switches to pure exploitation, choosing the machine with highest expected rewards.

The ϵ-decreasing algorithm consists of decreasing the value of ϵ with number of trials. The algorithm progresses from highly explorative approach in the beginning to a highly exploitative one towards the end.

CesaBianchi and Fisher (1998) found that ϵ-decreasing algorithm is theoretically efficient (with respect to regret). However empirical studies by Vermorel and Mohri (2005) do not seem to find any significant improvements with the variants of the algorithm when compared to the original ϵ-greedy algorithm.

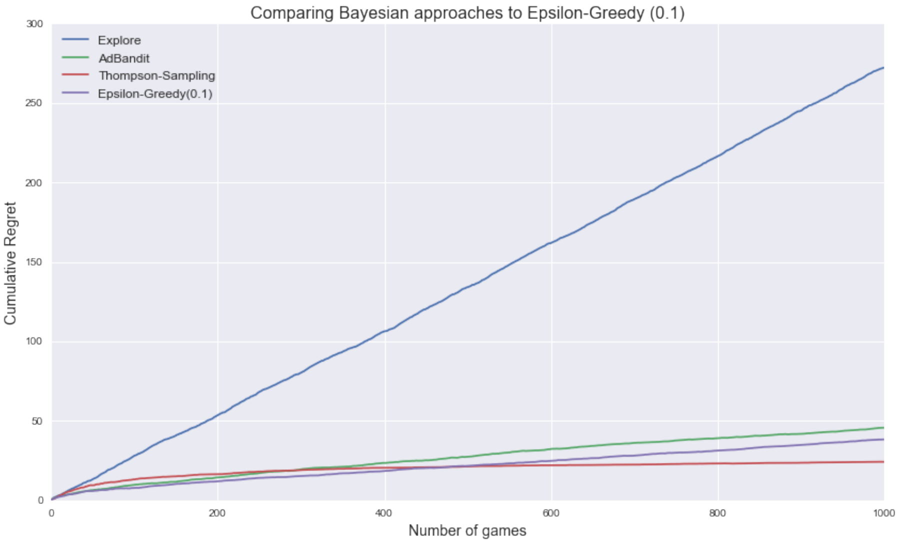


Figure 3: Cumulative Regret

References Bandit algorithms for Website Optimization by J. White algorithms for the multi-armed bandit problem by V. Kuleshov and D. Precup Multi-armed Bandit algorithms and Empirical Evaluation M. Mohri and J. Vermorel Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. machine Learning, 47((2-3)), 2002.

3.4 Upper Confidence Bounds

The other algorithms presented in this paper are similar in that they pay attention only to how much reward they've gotten from the machines. This means that they're likely to under-explore options whose initial experiences were not rewarding, even though they may not have enough data to be confident about those arms. One naive approach to solving this problem is to run the algorithm multiple times and take the average of the results. Another could be to run the algorithm for a very long time and hope the probabilistic nature will eventually correct for random variation in the reward distribution. Another approach is to use probability theory to bound our confidence in how good or bad each machine is. This is exactly what Upper Confidence Bounds does.

The upper confidence bound (UCB) family of algorithms selects the machine with the largest upper confidence bound at each round. This paper will only focus on UCB1, but note that there are many variants of the UCB family. The more times you play a machine, the tighter the confidence bounds become. So as the number of plays for each machine increases, the uncertainty decreases, and so does the width of the confidence bound.

We want to know with high probability that the true expected payoff of a play $\hat{\mu_i}$ is less than our prescribed upper bound:

$$\bar{\mu_i} + \sqrt{\frac{2ln(t)}{n_i}}$$

Where $\bar{\mu_i}$ is the average reward obtained from machine i and n_i is the number of times machine i has been played so far.

This upper bound is the sum of two terms, where:

1. the first term is the average reward
2. the second term is related to the one-sided confidence interval for the average reward according to the Chernoff-Hoeffding bounds

Recall that since rewards follow a Bernoulli distribution, we can apply Chernoff-Hoeffding to upper bound the probability that the sum of rewards from each machine deviates from its expected value:

$$P(Y + a < \mu) \leq e^{-2na^2}$$

This confidence bound grows with the total number of actions we have taken but shrinks with the number of times we have tried this particular action. This ensures each action is tried infinitely often but still balances exploration and exploitation. It can be shown that the regret for UCB1 grows with **ln n**, as witnessed below for the optimal machine.

Note that in addition to keeping track of our confidence in the estimated values of each machine, the UCB algorithm doesn't use randomness at all. Unlike the other algorithms in this paper (and in the literature) it's possible to know exactly how UCB will behave in any given situation. This can make it easier to reason about at times.

3.4.1 UCB1: algorithm

Assuming K machines:

1. Play each arm once
2. Observe rewards r_i, for $i = 1, ..., K$
3. Set $n_i = 1$, for $i = 1, ..., K$
4. set $\bar{\mu_i} = \frac{r_i}{n_i}$
5. For time $t = K + 1, ..., N$:
   (a) Play arm $\hat{i} = \underset{i}{\operatorname{argmax}}\ (\hat{\mu_i} + \sqrt{\frac{2ln(t)}{n_i}})$
   (b) Observe reward r
   (c) $r_{\hat{i}} = r_{\hat{i}} + r$
   (d) $n_{\hat{i}} = n_{\hat{i}} + 1$
   (e) update $\hat{\mu_i} = \frac{r_{\hat{i}}}{n_{\hat{i}}}$

Below we run UCB1 on the same dataset as above:

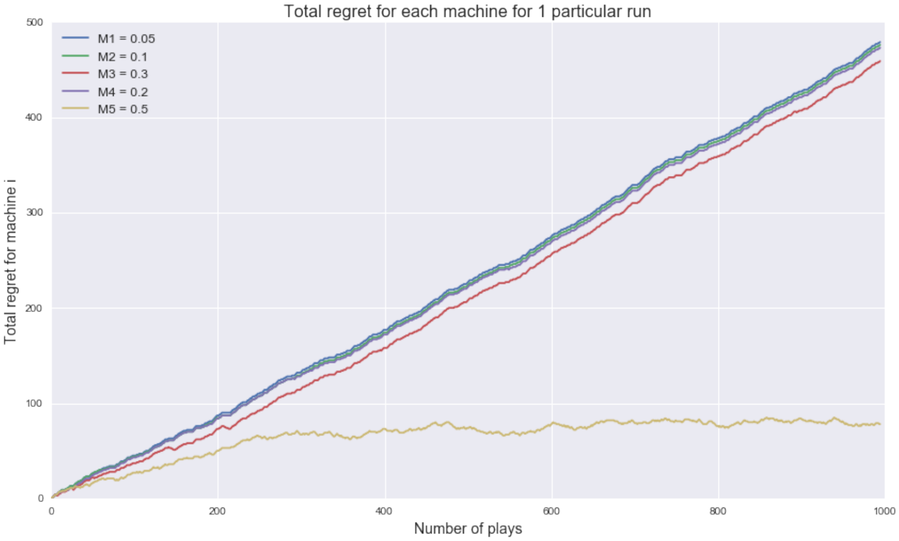


Figure 4: Cumulative for all 5 machines

Above we see that, as expected, Machine 5 has the lowest regret, since it has the highest θ.

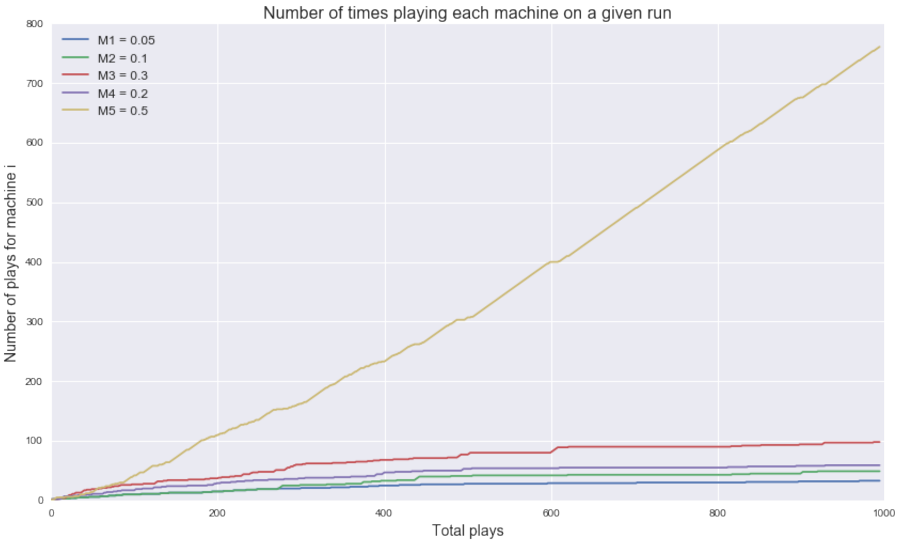


Figure 5: Switching between machines

As expected, machine 5 is played the most often, even though the worse off machines are still being played. Separation from the other machines occurs around 60 iterations. Also note that even at 300 runs, there isn't much separation between machines 1 and 2, given how similar their θ is.

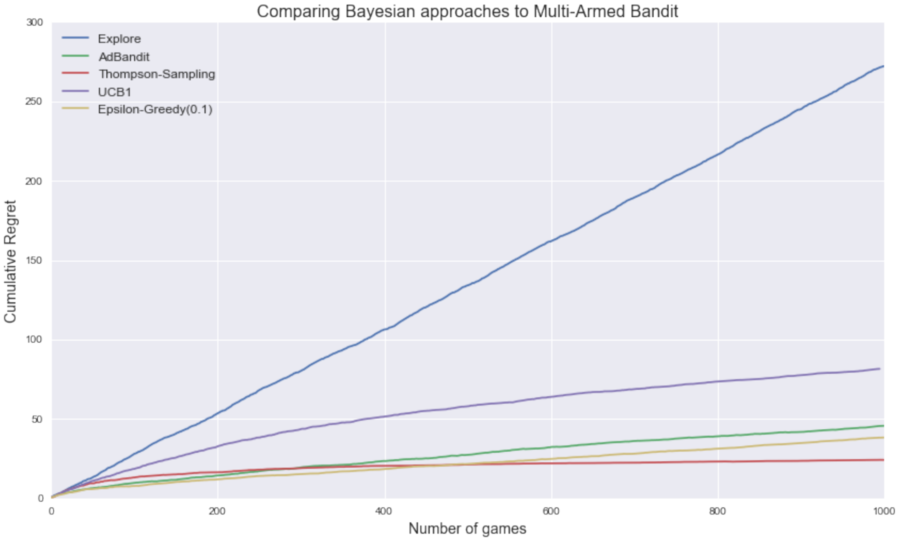


Figure 6: Comparing all 5 approaches

UCB1 does the worst in this particular run, compared to the other non-naive approaches. This suggests that while it provides nice theoretical guarantees, Bayesian heuristic approaches and even simple ones like ϵ-greedy can outperform it.

4 Continuous Reward

4.1 Gaussian Processes

We generalize our scheme to accomodate continuous rewards rather than Bernoulli. This scenario may be more intuitive in a slot machine or A/B testing analogy, where rewards may be continuous with respect to the dollars earned or lost. All methods we employ above, such as Epsilon Greedy, Thompson Sampling, Dynamic Programming, and UCB, can be easily applied to this continuous framework, but we present an interesting alternative method.

We model our reward function as a Gaussian Process (GP), specified by its mean and covariance function. We optimize an unknown reward function f. During each time step $t = 1, ..., T$, a Machine x is chosen at random and sampled for a noisy observation of the underlying reward function. More simply, we sample a reward from the latent reward function with added Gaussian noise.

4.2 Prediction

Assuming that the reward function is a sample from GP(0, k(x,x)) (as our prior), we can find the mean and covariance of the posterior predictive distribution as follows:

$$P(y_{t+1}|D_{1:t}, x_{t+1}) = N(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1}) + \sigma_{noise}^2)$$
$$\mu_t(x_{t+1}) = k^T[K + \sigma_{noise}^2 I]^{-1} y_{1:t}$$
$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - k^T[K + \sigma_{noise}^2 I]^{-1} k$$

We should choose the next point x where the mean is high (exploitation) and the variance is high (exploration). We balance this tradeoff by choosing an **acquisition function**, which guides the optimization by determining which x_{t+1} to observe next.

4.3 Acquisition Function

4.3.1 Probability of Improvement (PI)

$PI(x) = p(f(x) \geq \mu^+ + \xi) = \Phi(\frac{\mu(x)-\mu^+-\xi}{\sigma(x)})$ - μ^+ : The highest value of all the observed values so far
- ξ : Tuning parameter to adjust exploration vs. exploitation. We adjust the tuning parameters with a different number of machines to test efficiency.

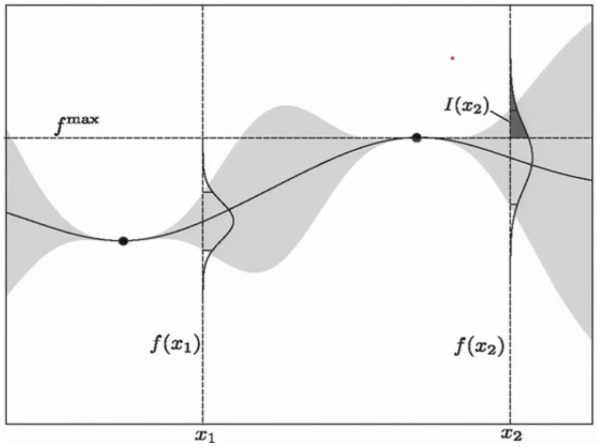


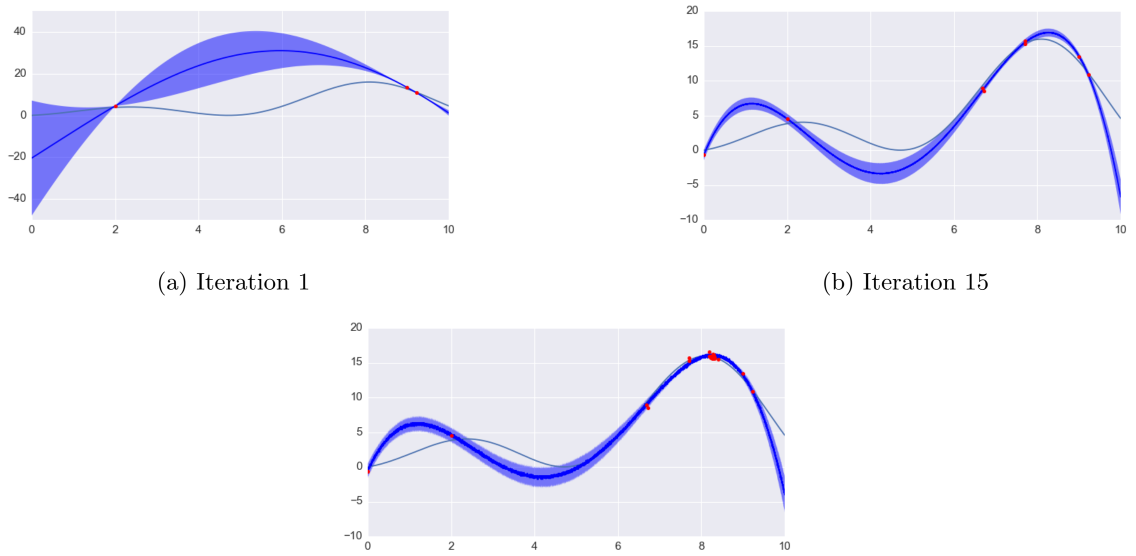
Figure 7: Probability of Improvement (PI)

The figure above demonstrates our acquisition function. We determine the probability of improvement as the CDF of the Prediction function evaluated at the highest observed value.

4.4 Empirical Results

We analyze our Gaussian Process scenario over a large number of Machines in a continuous reward setup, creating a latent reward function and sampling observations based on the function. Note that as the number of iterations increase, our GP closer resembles the 'true' reward function, and ultimately the utility function chooses the best point every time.

In Figure 6, the latent reward function is also plotted, and we can see that over time, our observed values tend to cluster close to the highest point (ie: the best machine), and the overall fit matching close to the true function. We also compare the cumulative Reward and Regret in the GP method.



(a) Iteration 1

(b) Iteration 15

(c) Iteration 30

Figure 8: GP Fit evolves as we sample rewards and iterate back into prediction function.

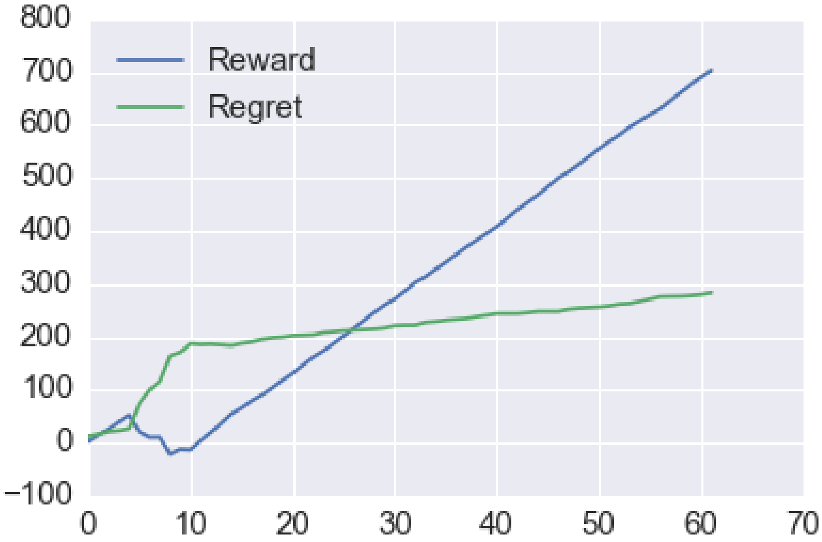


Figure 9: Cumulative Regret and Reward

We can see that in the first 10 iterations, the GP method is exploring high variance areas of our reward function, resulting in a lower reward and higher regret, but after this point, it settles upon the best options. From there, the regret flattens out and the reward increases linearly while exploiting the best machines.

5 Conclusion

6 References