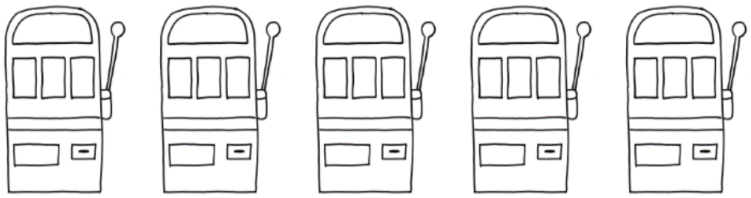
Multi-Armed Bandits

John Adenigbagbe, Adam Boustani, Jon Cheng, Uathesh Ranjan, Dev Shah

M2R - Group 47



Project Supervisor - Dr Ciara M Pike-Burke

Acknowledgements

We would like to give special recognition to Dr Ciara Pike-Burke for creating this project and for guiding us throughout our journey. She has introduced and helped us to understand the key concepts surrounding multi-armed bandits. Her advice has helped shape our research.

We would also like to thank Imperial College London for providing us with this opportunity and allowing us to research a topic we are all passionate about.

Abstract

The multi-armed bandit (MAB) problem is a statistics problem where given a number of slot machines, or 'arms', with unknown reward distributions, we aim to find the optimal strategy of playing the machines in order to maximise the cumulative rewards over time.

We begin by introducing the MAB problem and its background and then investigate the exploration-exploitation trade-off, which emphasises the importance of balancing between acquiring new information about each arm and exploiting the currently perceived best arm.

We then implement three basic MAB algorithms: Round Robin, Follow the Leader, and Epsilon-Greedy. We compare these different approaches to the MAB problem and evaluate their effectiveness on different numbers of machines as well as on different reward distributions (Bernoulli, Normal, and Uniform).

These findings allow us to introduce more advanced algorithms, UCB1 and Thompson Sampling, which find a better balance between exploration and exploitation, and we discuss the assumptions necessary for implementation.

Finally, we analyse variations to the MAB problem such as adversarial bandits and contextual bandits, and explore how these provide us with applications across various domains.

Contents

1 Introduction

The multi-armed bandit (MAB) problem is a statistical concept that originated as a gambling game involving slot machines, otherwise known as one-armed bandits. The challenge for players was to decide which levers to pull to maximise their total winnings, given limited information about the probabilities associated with each lever. We will refer to each slot machine as an 'arm' and its payoff as the 'reward'.

Mathematicians began formalising the multi-armed bandit problem in the early 1950s. They wanted to find an optimal strategy for selecting levers to maximise long-term rewards. Their research introduced the exploration-exploitation trade-off, which at the time involved deciding whether to try out a new lever to gather more information (exploration) or to stick with a known good one (exploitation).

This problem is a classic example of reinforcement learning, where we determine the optimal decision-making process based on observed information we have collected so far. Understanding this problem is crucial as it offers valuable insights into how we can efficiently allocate limited resources in order to maximise long-term gains. Over time, a range of MAB algorithms were developed, providing varying strategies with differing strengths and weaknesses.

In this paper, we discuss the trade-offs between algorithms and evaluate their effectiveness on different reward distributions, considering the importance of both exploration and exploitation. We also analyse variations to the MAB problem such as adversarial bandits and contextual bandits, exploring how these are applicable to real-world scenarios.

2 Problem Formulation

The objective of the MAB problem is to maximise long-term cumulative rewards given a set of slot machines with unknown reward distributions. However, as rewards can vary between problem instances, in practice we determine how to minimise cumulative regret. The regret in a round is the difference between the mean of the optimal arm and the mean of the arm played in that round [1], and cumulative regret is the sum of the regret incurred up to round T.

For each round $t = 1, ..., T$ in the MAB problem we:

- Select an arm $J_t \in \{1, ..., K\}$ to play,
- Receive reward $X_{J_t,t} \sim \nu_{J_t}$,
- Update the history, $H_t = H_{t-1} \cup \{J_t, X_{J_t,t}\}$.

We also define the following:

- The number of times arm j has been played:
  $N_j(t) = \sum_{s=1}^{t} \mathbb{I}\{J_s = j\}$.
- The empirical average reward of arm j up to round t:
  $\hat{\mu}_{j,t} = \frac{1}{N_j(t)} \sum_{s=1}^{t} X_{j,s}\mathbb{I}\{J_s = j\}$

We define the regret after T rounds, for an algorithm π, as:

- $R_T(\pi) = \sum_{t=1}^{T} \mu^* - \mu_{J_t}$

We often assume that the reward distribution is σ-sub-Gaussian. A random variable X is said to be from a σ-sub-Gaussian distribution with mean μ if for all $\lambda \in \mathbb{R}$:

- $\mathbb{E}[e^{\lambda(X-\mu)}] \leq \exp(\frac{\lambda^2\sigma^2}{2})$

3 Exploration vs Exploitation

In the MAB problem, the challenge lies in finding the optimal balance between exploration and exploitation. Early in this process, exploration is crucial to estimating the true means of the arms accurately. As more information is gathered, the balance shifts towards exploitation to maximise immediate rewards based on current knowledge.

Exploration:

- Exploration involves trying out different actions to gather information about their rewards.
- The goal is to discover the true mean reward of each arm.
- During exploration, sub-optimal arms are intentionally pulled to estimate their rewards accurately.
- The aim is to reduce uncertainty and improve the accuracy of reward estimates for all arms.
- Exploration helps in making informed decisions in the long run but may result in smaller short-term rewards.

Exploitation:

- Exploitation involves selecting the arm that is expected to have the highest reward based on current knowledge.
- The goal of exploitation is to maximise immediate rewards by favouring arms with higher estimated means.
- Exploitation is driven by the belief that pulling the arm with the highest estimated mean so far will produce the best outcome.
- It prioritises taking actions that have been observed to perform well in the past.
- Exploitation is focused on maximising the short-term reward and taking advantage of the current knowledge.
- However, relying solely on exploitation may lead to missed opportunities for higher rewards if better arms remain unexplored.

The exploration-exploitation trade-off is a central concept in solving the MAB problem and requires careful consideration to achieve optimal long-term performance. This can be seen by analysing the outcomes of the "Round Robin" and "Follow the Leader" algorithms which rely purely on exploration and exploitation respectively, as well as comparing these with other algorithms which strive to find a balance.

4 Basic Bandit Algorithms

In this section, we discuss the more basic approaches to the MAB problem, which use the Round Robin algorithm, the Follow the Leader algorithm and the Epsilon-Greedy algorithm. These provide a basis for the other more advanced algorithms, which we discuss later. Note that we refer to an arm with a given reward distribution X as an X arm.

For the plots in this section and for a given distribution, we fix the arms used in each algorithm. For example, the same 5 Bernoulli arms are used in the Round Robin, Follow the Leader, and Epsilon-Greedy algorithms, showcased in the first plot in each subsection. To ensure consistency in our analysis between distributions, we evenly space the means of each arm between 0 (inclusive) and 1 (exclusive) in every plot. We run each algorithm over 1000 rounds, and the cumulative regret is averaged over 1000 games. The shaded regions in each plot represent the (non-negative) ± 1 standard deviation of the cumulative regret, arising from the 1000 games played with each algorithm.

In all the figures below, we demonstrate how the cumulative regret varies with both the reward distributions and the number of arms. We consider 5 arms, 25 arms and 50 arms, and in our graphs we indicate Bernoulli arms in blue, Normal arms in green and Uniform arms in red.

4.1 Round Robin Algorithm

The Round Robin algorithm is based purely on exploration. We play each arm one at a time and the rewards outputted do not affect how we proceed with the problem. As a result, we do not need to make any assumptions on the distributions of the rewards for this algorithm.

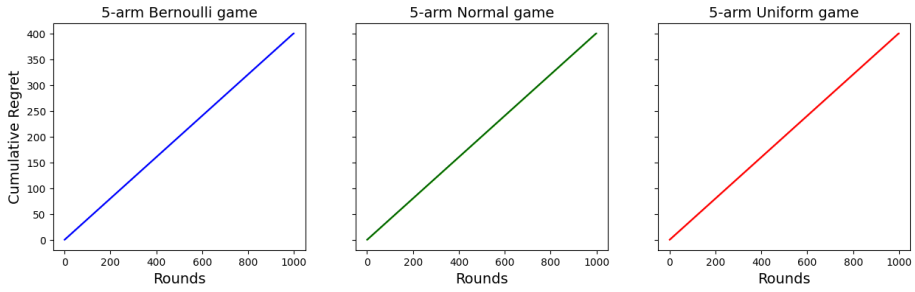


Figure 1: 5-arm Round Robin

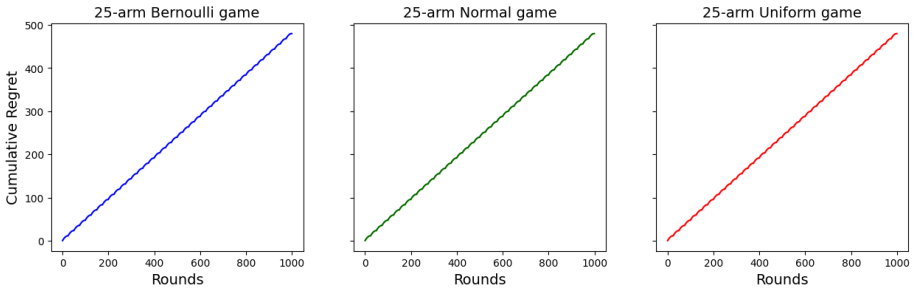


Figure 2: 25-arm Round Robin

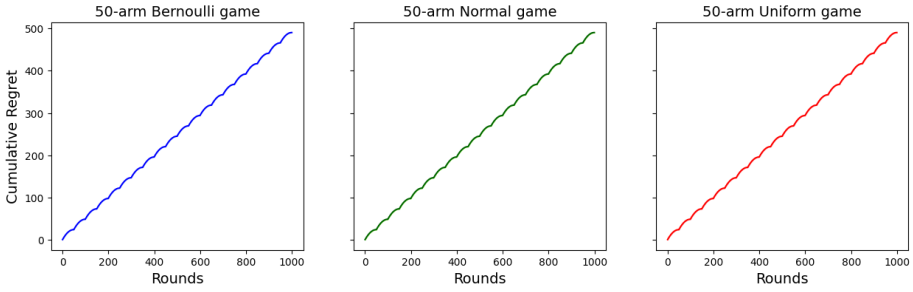


Figure 3: 50-arm Round Robin

Figures 1, 2, and 3 show a positive linear relationship between the number of rounds and cumulative regret, regardless of the reward distribution. As we play each arm one at a time in this strategy, in every K rounds, we select the optimal arm exactly once. This explains the regular bumps in the graphs. The variance of the cumulative regret is 0 since we play all the arms in every K rounds, so cumulative regret increases by the same amount in each round, when averaging over 1000 games. In each figure, we get the same cumulative regret, independent of the distribution used, due to the assumption that the means of each are are evenly spaced between 0 and 1.

As the number of rounds tends to infinity, we do not see any convergence in cumulative regret. Thus, we can dismiss 'Round Robin' as a viable strategy.

4.2 Follow the Leader Algorithm

The Follow the Leader algorithm is based purely on exploitation. We play each of the arms once in order to obtain initial estimates for their mean rewards. Then the arm with the largest mean is played and after receiving a reward, we update its empirical mean. We continue to play this arm until its mean falls below the mean of another, and at this point we switch to this other arm which now has the highest mean. As we play arms according to the samples collected, this strategy does not require any assumptions on the reward distributions.

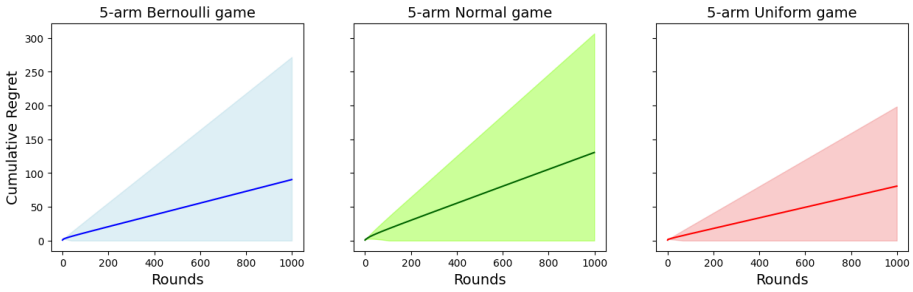


Figure 4: 5-arm Follow the Leader

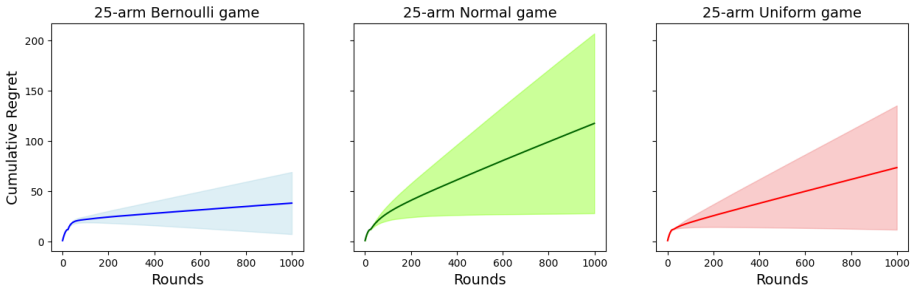


Figure 5: 25-arm Follow the Leader

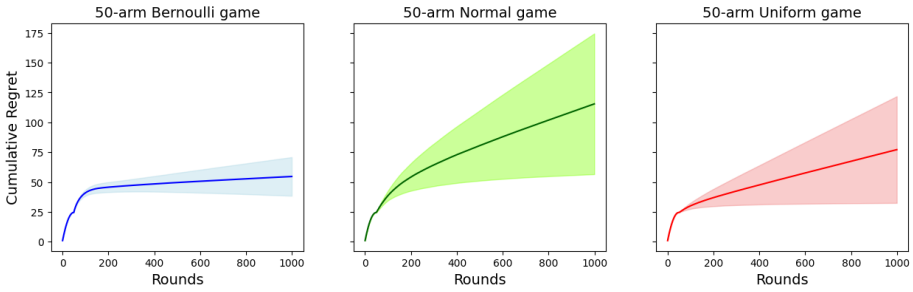


Figure 6: 50-arm Follow the Leader

Early on, the cumulative regret rises sharply because we are pulling each arm once to collect information, so clearly we will obtain quite a few sub-optimal outputs. After this stage, the cumulative regret still increases linearly, but with a shallower gradient. Variance also arises since our strategy can now differ in terms of the order in which we play the arms - it increases especially for Normal arms and Uniform arms because the reward is not restricted to 1, so the empirical mean isn't restricted to $[0, 1]$, and this leads to a variation in how often the optimal arm is played.

Follow the Leader may seem like a good algorithm, but the lack of exploration is a major flaw. As each arm was only selected once initially, there is no way to gauge an accurate estimate of its true mean reward. Moreover, suppose by chance we obtain higher rewards on some sub-optimal arms. This algorithm would then lock onto them and consequently, we would not play certain arms that could provide a higher expected reward, leading to a larger variance in cumulative regret shown in the graphs. Hence, Follow the Leader does not care for the long-term plays of a game, choosing to ineffectively focus on the locally optimal choice instead.

4.3 Epsilon-Greedy Algorithm

An improvement to Follow the Leader would be to incorporate more exploration while also exploiting. The Epsilon-Greedy algorithm achieves this; while we still play arms which seem to have a higher mean reward, we maintain an open mind to the other ones. We play each arm once and for every subsequent round, there is a probability ϵ that we choose an arm at random. Otherwise, we play arm $J_t = \arg\max_{1 \leq j \leq K} \hat{\mu}_{j,t}$.

In the graphs below we set $\epsilon_n = \frac{1}{n}$, where n is the number of rounds played so far, after having tried each arm once initially. We are linearly decaying the value of ϵ, i.e. $\epsilon \to 0$, so that as we more accurately determine the expected reward of each arm, we reduce the amount of exploration.

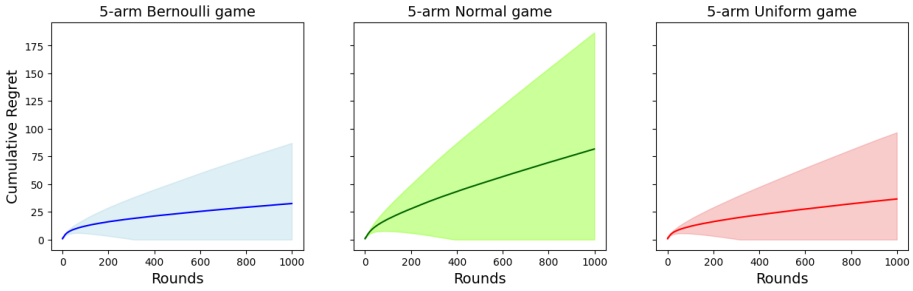


Figure 7: 5-arm Epsilon-Greedy

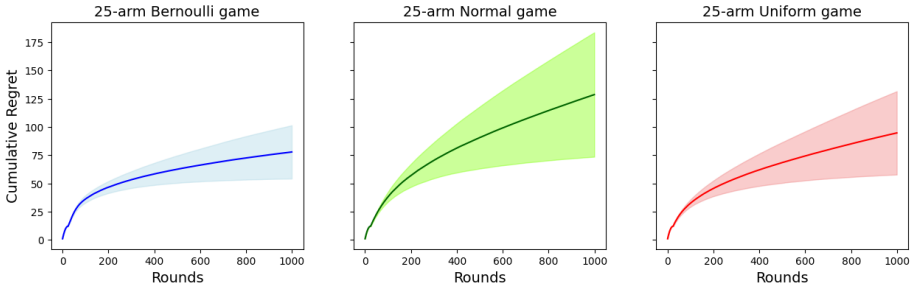


Figure 8: 25-arm Epsilon-Greedy

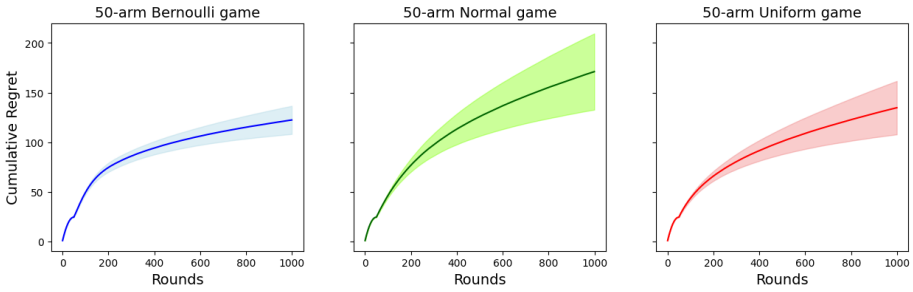


Figure 9: 50-arm Epsilon-Greedy

Figures 7, 8, and 9 portray a more logarithmic relationship between cumulative regret and the number of rounds for the Epsilon-Greedy algorithm using different distributions.

Like the above algorithms, we see cumulative regret generally increasing but here it is less linear. This is due to the added exploration, which restrains the algorithm from fully using the information gained in each round as it is obtained. As we continue to play more rounds, the algorithm will explore less and we will stick with a relatively optimal arm, illustrated by the cumulative regret increasing at a decreasing rate.

4.4 Comparing Algorithms

Analysing the three basic bandit algorithms, we find that Follow the Leader finds what it thinks is the best arm and begins playing it more quickly than the other two algorithms.

With a few arms, the Epsilon-Greedy algorithm seems to be the superior algorithm as it has lower regret compared to the Follow the Leader algorithm. However, when a larger number of arms are used, Follow the Leader seems to have lower regret. This may be because with more arms, the Epsilon-Greedy algorithm still randomly tries other arms which may lead to high regret. The standard deviation for the Epsilon-Greedy algorithm is lower than that of Follow the Leader since there is more exploration in the former algorithm, resulting in lower variance in the regret distribution.

Despite outperforming the other two algorithms when fewer arms are used, Epsilon-Greedy is still not ideal because its exploration phase is non-adaptive. In this phase, the history of observed rewards is not used to inform how an arm is selected; instead, a random arm is chosen to be played. As a result, even if the algorithm has enough information to infer that an arm is sub-optimal, this arm isn't ruled out, and can continue to be selected during the exploration phase, increasing cumulative regret.

5 Advanced Bandit Algorithms

Having analysed the 3 previous basic ones, we take the best parts of each of them to try and optimise the trade-off between exploration and exploitation to maximise the reward. These advanced bandit algorithms are discussed below.

5.1 UCB1 Algorithm

The UCB1 algorithm is centred on the Optimism Principle, which assumes that 'each arm is as good as it can possibly be given the observations so far' [2]. This translates to constructing overestimates for the means of each arm through the information we have collected in the form of upper confidence bounds (UCBs). We then play the arm with the largest UCB, which means we have the potential to play arms that may not have the highest empirical mean. This is us behaving optimistically: we can give rarely used arms the benefit of the doubt, i.e. we explore arms for which we haven't confidently ascertained the expected reward.

We make this bound precise via Hoeffding's Inequality [3]: Let $X_1, \ldots, X_n$ be n i.i.d. samples from a σ-sub-Gaussian distribution with expected value μ and let $\hat{\mu}^n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Then for any $\epsilon > 0$,

$$\mathbb{P}(|\hat{\mu}^n - \mu| \geq \epsilon) \leq 2\exp\{-\frac{n\epsilon^2}{2\sigma^2}\}$$

If we choose some $\delta > 0$ and set it equal to the right-hand side of the inequality, we can then deduce that in any round t, with probability greater than 1 - δ,

$$|\hat{\mu}_{j,t} - \mu_j| \leq \sqrt{\frac{2\sigma^2 \log(2/\delta)}{N_j(t)}}$$

where $\hat{\mu}_{j,t}$ is the empirical average reward and μ_j is the true mean of arm j.

Clearly we can define an upper bound for the true mean by

$$\mathrm{UCB}_t(j) = \hat{\mu}_{j,t} + \sqrt{\frac{2\sigma^2\ \log(1/\delta)}{N_j(t)}}$$

where this δ we have specified is our confidence parameter, indicating how tight we want our upper bound to be (typically $1/(KT^2)$). The reason we have $1/\delta$ rather than $2/\delta$ is because we use one tail in practice - this upper bound was derived from the one-sided form of Hoeffding's Inequality.

To start off the algorithm, we play each arm once and with the values we obtain, calculate their upper bounds. For the subsequent rounds, we want to play the arm with the largest upper bound. Then having received a reward, we update both the arm's empirical mean and UCB and continue. The algorithm is depicted in Algorithm 1.

Algorithm 1: UCB1 algorithm

Input: arms, T (number of rounds), δ (confidence parameter)

for $t = 1, ..., K$ **do**
  Take a sample $X_{t,t}$ from each arm, and update its empirical mean reward $\hat{\mu}_{t,t}$ and UCB.
end
for $t = K + 1, ..., T$ **do**
  Choose the arm $J_t = \arg\max_{1 \leq j \leq K} \text{UCB}_{t-1}(j)$ and receive a reward $X_{J_t,t}$ from this arm;
  Update this arm's empirical mean reward $\hat{\mu}_{J_t,t}$ and UCB.
end

By looking at the formula for $\text{UCB}_t(j)$, we can observe the trade-off quite naturally. If the UCB of an arm is high, either $\hat{\mu}_{j,t}$ is large (i.e. we are exploiting what seems to be a successful arm) or $\sqrt{\frac{2\sigma^2 \log(1/\delta)}{N_j(t)}}$ is large, which occurs when $N_j(t)$ is small (i.e. we have barely played that arm and so we explore by trying it out). And as we play more and more rounds, this latter exploration term will become redundant because as $\text{T} \to \infty$, $\frac{\log n}{n} \to 0$, so in the end we just exploit.

Despite requiring a sub-Gaussian assumption on the rewards of the arms, UCB1 is a more favourable algorithm than those previously explored, with its cumulative regret eventually achieving a logarithmic shape. In fact, it is bounded above by $\mathcal{O}(\sqrt{KT \log T})$, which is better than the algorithms whose regret increased linearly. However, in practice with more arms, UCB1 does increasingly worse as we will soon see.

5.2 Thompson Sampling

Now we will take a Bayesian approach to the MAB problem. The setting is essentially the same, but initially we choose a prior distribution (from the same family) for each of the K arms, and subsequently sample from them to compute posteriors. Intuitively, the idea of Thompson sampling is to choose an arm to play according to the probability that it is optimal [4].

We assume that the rewards of the arms are 'nicely' distributed in the sense that they have a conjugate prior that can be used as the initial prior, which simplifies the implementation of Thompson sampling. So for Bernoulli arms we choose a Beta prior; for Normal arms we choose a Gaussian prior; and for Uniform arms we choose a Pareto Type I prior [1].

Algorithm 2: Thompson sampling algorithm

Input: Prior distributions $\pi_{1,0}, \ldots, \pi_{K,0}$ of each arm, T (number of rounds);

for $t = 1, \ldots, T$ **do**
  Take a sample $\tilde{\mu_j} \sim \pi_{j,t}$ from each arm;
  Choose the arm $J_t = \arg\max_{1 \leq j \leq K} \tilde{\mu_j}$ and take a sample $X_{J_t,t}$ from this arm;
  Update this arm's posterior distribution $\pi_{J_t,t}$ using the sample $X_{J_t,t}$.
end

The algorithm begins by sampling once from the posterior distribution of each arm (so for the first time we run the algorithm, we sample from the initial prior we set to be the same across all arms). Then we play the arm with the highest sample and update the posterior of the played arm with the new observation (i.e. perform a Bayesian update), having calculated the new parameters for the posterior distribution. Note that we maintain the distributions of the other arms. We continue this process of sampling and performing Bayesian updates, so that rather than just estimating the expected reward of each arm as we play, we figure out a probability model for its reward and sample accordingly.

If the sample we obtain is large, then it could be the case that the posterior mean is large. Hence, we will be exploiting. Alternatively, it could be due to the posterior variance being large, which would be the case for less tested arms (and thus we are exploring). This is because taking more samples from the same arm will decrease its posterior variance as we become more confident in its expected

[1] If X is a Pareto Type I distributed random variable, then for the scale parameter $x_m > 0$ (the minimum possible value of X), and the shape parameter $\alpha > 0$, X has the pdf

$$f_X(x) = \begin{cases} \frac{\alpha x_m^\alpha}{x^{\alpha+1}}, & x \geq x_m \\ 0, & x < x_m \end{cases} \text{[5]}.$$

value. So for less-tested arms, there is a wider scope for a higher reward. As a result, Thompson sampling achieves a lovely trade-off between exploitation and exploration.

Although UCB1 requires the reward distribution to be σ-sub-gaussian, the distribution does not need to be known, whereas for Thompson sampling, as we use a Bayesian approach, we assume that the reward distribution is known. Moreover, while both advanced algorithms achieve the same theoretical bound for cumulative regret, Thompson sampling is used in practice because it performs much better empirically, which will be illustrated in the next section.

5.3 Visualising UCB1 and Thompson Sampling

In this section, we fix the arms used in each figure, and compare the UCB1 and Thompson sampling algorithms over 5000 rounds, with cumulative regret averaged over 200 games.

In Figure 10, we use 3 Bernoulli arms, with probabilities of success 0.9, 0.1 and 0.1, and in Figure 11, we also use 3 Bernoulli arms, with probabilities of success 0.15, 0.1 and 0.1.

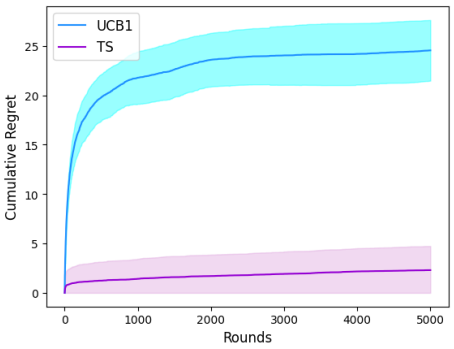


Figure 10: Optimal arm mean = 0.9.

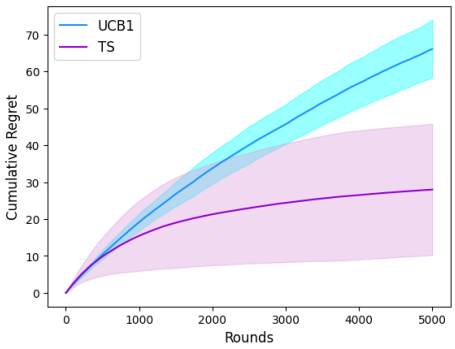


Figure 11: Optimal arm mean = 0.15.

In both figures, we observe that Thompson sampling is the preferred algorithm, as we see lower cumulative regret than when using UCB1. In Figure 10, as the difference between the mean of the optimal arm and the mean of the other arms is larger than in Figure 11, cumulative regret flattens out within 2000 rounds, indicating that both algorithms have found the best arm to play. In Figure 11, as there is a smaller difference in the means of each arm, more rounds are necessary for regret to stabilise. Focusing on UCB1, we see that a larger difference in the mean of the optimal arm to that of the second-best arm, although inducing larger regret on a per round basis when the optimal arm isn't played, produces the optimal arm much more quickly, as it takes fewer rounds for the UCB of the optimal arm to be sufficiently high such that we don't play the second-best arm again.

In the following figures, the means of the arms are evenly spaced between 0 and 5. In Figure 12, 5 Normal arms are used, with fixed variance 1, and in Thompson sampling we use the prior hyperparameters $\mu = 2.5$ and $\sigma^2 = 1$. In Figure 13, 5 Uniform arms are used and in Thompson sampling we use the prior hyperparameters $x_m = 1$ and $\alpha = 1$.

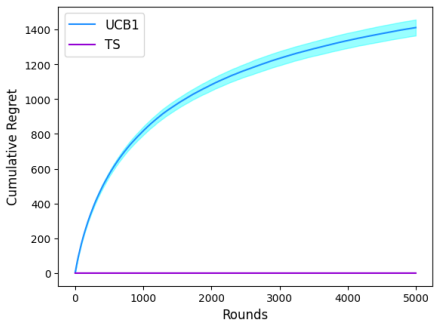


Figure 12: 5 Normal arms.

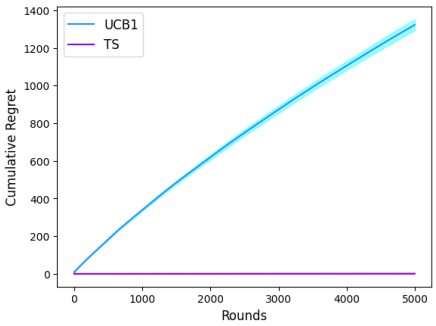


Figure 13: 5 Uniform arms.

Again, we see that Thompson sampling out-performs UCB1, where with extensive experimentation, we found suitable prior hyperparameters to use. We see that cumulative regret for UCB1 flattens out in less rounds for Normal arms than for Uniform arms. As the UCB is linearly proportional to the standard deviation of each arm, less rounds are required to determine the arm with the highest mean reward in the case of Normal arms, as they have variance 1, than for Uniform arms, which have variance $\frac{(b-a)^2}{12}$ that depends on the maximum of the uniform distribution.

The above plots suggest that Thompson sampling is preferred to UCB1 in practice. If the reward distribution of an arm is known, and has a conjugate prior, Thompson sampling accurately approximates this distribution using the conjugate prior and the rewards observed in each round, whereas UCB1 compares the UCB of one arm to the UCB of all other arms, a much slower process that produces greater cumulative regret. However, the reward distribution of an arm may not be known, or this distribution may have an intractable posterior, requiring numerical methods to calculate, increasing computational cost. In these instances, UCB1 would be suitable to use, as the reward distribution of an arm need not be known to find a confidence interval for the empirical mean reward.

6 Extensions, Variations and Applications of the Multi-Armed Bandit Problem

"The simple k-armed bandit problem performs sub-optimally by its design in the advertising context"

Giuseppe Burtini, Jason Loeppky, Ramon Lawrence

Is the simple multi-armed bandit problem truly applicable to real-world scenarios? If we consider the setting of K slot machines in a casino, can one really reasonably assume that the reward distributions of the machines are truly fixed, and randomly sampled? Or perhaps we could extend the game further - what action should one take if asked to choose multiple arms per round?

In reality, there is a high likelihood of there being other forces at play. Casinos - and indeed the real world - are far too complex to safely make such assumptions. This section explores more applicable variations of the simple bandit problem, and a selection of algorithms for each.

6.1 Adversarial Bandits

As suggested by the name, the adversarial bandit problem represents the situation where an 'adversary' is actively plotting against the player. The adversary chooses the reward of each arm for each round, with the intention of maximising the player's regret.

This is in fact a very common scenario in the real world. One example is in cybersecurity, where the adversary could be a cyber-attacker. Using an adversarial bandit problem, we can test the effectiveness of defence systems and optimise them against all sorts of different attack strategies.

The most notable difference here is that in the classical scenario, there was a fixed distribution from which the rewards were sampled from. However, now we have an adversary choosing the rewards. Any algorithms that we use can no longer rely on statistical assumptions made about rewards distributions.

6.1.1 The Adversary

Different kinds of adversaries can be defined by placing constraints on an adversary's knowledge and actions. We will focus on a 'standard' adversary that:

- knows the player's algorithm,

- generates a sequence of rewards $X_{k_1}, X_{k_2}, ..., X_{k_t}$ for each round, for each arm k, before the game has begun, using knowledge of the algorithm,
- does not know what arm the player chooses in each round.

It is important to note that other kinds of adversaries can be defined. For example, one could use a function of a player's previous choices to determine the rewards for each arm for the next round.

Immediately we can see this variation essentially breaks down high-exploitation algorithms, such as the UCB1 algorithm. Consider our standard adversary as defined initially. Using knowledge of the UCB, the adversary could give a very low upper confidence bound for a certain arm during the exploration stage. Once said arm has been eliminated by the algorithm, it would then assign maximal rewards to this arm for the remainder of the game, thus maximising the cumulative regret gained by the player.

6.1.2 The Exp3 Algorithm

We now look at the Exp3 Algorithm, which works relatively well within the adversarial bandit problem. This algorithm strikes a good balance between exploration and exploitation, and uses weights to determine the course of action, of which are exponentially updated as more information is gathered.

To begin, the user initialises an exploration parameter $\gamma \in [0, 1]$. This is known as an *egalitarianism factor*. γ represents the proportion of rounds that the algorithm spends on random selection, and $(1 - \gamma)$ gives the proportion of rounds spent on weighted exploration and exploitation [6].

The user then selects a *learning rate parameter*, $\eta \in [0, 1]$. This represents the proportion of rounds in which the algorithm can apply a weight update formula - in other words how quickly the algorithm "learns". η is often given a default value of $\sqrt{\frac{2 \log K}{TK}}$ (where T is the total number of rounds to be played), allowing it to decrease as the number of rounds or arms increases. This promotes exploration early on, then promotes a shift towards exploitation as more data is gathered.

The algorithm then initialises a weight vector, with each element corresponding to a weight assigned to each arm in the game. We start with every weight being 1, and these are updated accordingly using the weight update formula and η as the game progresses (detailed below).

$$w_j(t+1) = w_j(t) * \exp[\eta \frac{x(t)}{Kp_j(t)}]$$

where:

- $w_j(t)$ gives the weight of arm j in round t,

- $x(t)$ gives the observed reward of the action in round t,
- K is the total number of arms,
- $p_j(t)$ gives the probability of selecting arm j in round t, of which the formula is given below:

$$p_j(t) = (1-\gamma)\frac{w_j(t)}{\sum_{i=1}^{K} w_i(t)} + \frac{\gamma}{K}.$$

The algorithm uses the probabilities to choose an arm to play, observes the reward, then accordingly updates the weight vector.

The Exp3 algorithm uses an effective exponential weight update formula, allowing it to adjust arm probabilities accordingly when observing the rewards provided by an adversary. It maintains a good rate of exploration and adapts well to changes in rewards. One can also show that the expected regret of the algorithm is bounded above by $\sqrt{2}\sqrt{Tk\log k}$ [7] [8].

However this algorithm does come with its disadvantages, with the main one being that it does not capture all possible scenarios. Examples include additional contextual information (we will see more about this in the next section), or if the arms have any sort of underlying structure (Exp3 does not consider if the rewards gained from one arm affect the rewards of any other arms). Furthermore, in certain scenarios it may incur much more regret than other algorithms, especially for the case where there is no adversary. As long as $\gamma > 0$, the algorithm will always be exploring bad options - a direct result of the "pessimistic" view it has.

6.2 Contextual Bandits

Consider a casino, in which we have a row of K slot machines. It is natural to ask whether other features outside of the distributions affect the rewards. Do the colours on the slot machines have any effect on the reward of the machine? Do certain machines pay more on certain days of the week? It could very well be the case that the payout is higher on a Monday as part of a drive to increase footfall in the casino on quiet days.

6.2.1 Contexts

The contextual bandit variation asks questions like these. We begin to consider contextual information that the rewards are dependent on. This makes the MAB problem much more applicable to the real world, as it is very rare to find a situation in which no external factors affect our arms.

One simple example to motivate contextual bandits is the case of online advertising. If one simply tried selecting adverts across a wide range of interests it

is likely that the choices would be meaningless to the user. One must tailor their choices to the user, and take into account information such as recent browser history, cookies and user demographics of the website - these are examples of "contexts" that we consider in the contextual variation.

Note that while there is very rarely no context affecting the reward distributions, it is not rare for the contexts to be completely unknown.

We can sort context into two types [9], using the example of a clinical trial to illustrate these:

- Arm Context - properties of the selection variable, such as dosage of a treatment;
- World Context - properties of the environment, such as any allergies a patient may have.

6.2.2 Algorithms for Contextual Bandits - LinUCB and Exp4

The LinUCB algorithm is an extension of the UCB algorithm, for contextual bandits. The general idea of the algorithm is the same; we still select the arm that gives the largest upper confidence bound, and eliminate others. The difference is that the LinUCB algorithm assumes a linear relationship (using linear regression models) between the contexts (which can be represented in a vector) and expected reward of each arm. This linear relationship plays a part in estimating the confidence intervals for each arm.

LinUCB shares the advantages of the UCB algorithm, and in additon is able to take contextual information into account, leading to more informed decision making. Furthermore, the use of linear regression makes the LinUCB less computationally complex than other contextual bandit algorithms. This allows for more efficient use in larger-scale problems.

The main disadvantages are that it performs poorly in the face of adversaries, and the assumption of a linear relationship between contexts and rewards. More recent work in algorithms such as *Banditron* and *Neural Bandit* is in place for the study of non-linear contextual bandits. LinUCB also assumes the contexts remain stationary, which is not a likely situation in the real world - one would expect a casino to attempt to incur losses upon a player after consistent wins.

The Exp4 algorithm is an adaptation of the Exp3 algorithm. Another "level" of exploration and exploitation is added on top to consider external factors that may affect reward distributions.

Exp4 shares the advantages of Exp3, with the addition of being able to factor in context in its decision making process. One significant disadvantage, however, is the computational cost of running such an algorithm. Exp4's running time

scales linearly with the number of contextual options available to it, thus is not the best option when considering complicated scenarios with a lot of contextual information.

6.3 Bandits with Multiple Plays

Komiyama, Honda and Nakagawa introduce this simple variation of the MAB problem in their work [10]. Instead of choosing one arm per round, we select $m \in \{1, 2, ..., K\}$ arms simultaneously per round. It is more applicable to scenarios where multiple selections are required - for example when browsing a certain product on a website, multiple recommendations of similar products are often displayed below.

The algorithm introduced by Komiyama et al. is an extension of Thompson sampling, called multiple play Thompson sampling (MPTS). The algorithm is almost identical to Thompson sampling, however, it now selects the best m arms, instead of just one. MPTS shares the strong benefits of Thompson sampling, namely achieving optimal regret and requiring minimal computation.

However, it can indeed be further improved - there exists another algorithm called IMPTS (improved multiple play Thompson sampling), which has been shown to have an advantage over MPTS. IMPTS instead chooses the first $m-1$ arms with the highest empirical expectations, then uses Thompson sampling to select the final arm. This improvement turns out to reduce regret, by incorporating more exploitation in selecting the first $m-1$ arms. It removes the possibility of drawing multiple sub-optimal arms, which can come about from applying Thompson sampling repeatedly.

7 Conclusion

Our aspiration for this project was to implement different multi-armed bandit algorithms and evaluate their effectiveness. We have analysed the fundamental trade-off between exploration and exploitation and how different strategies go about finding the right balance. We discussed the Round Robin algorithm, a pure exploration algorithm that produced the greatest cumulative regret, the Follow the Leader algorithm, a pure exploitation algorithm with linear cumulative regret, and the Epsilon-Greedy algorithm, a combination of the two that displays logarithmic cumulative regret. We also observed that whilst UCB1 and Thompson sampling have the same problem independent regret bound, Thompson sampling performs much better in practice.

Finally, we looked at variations of the multi-armed bandit problem and their real-world applications. We examined adversarial bandits and contextual bandits, exploring different algorithms of each, as well as bandits with multiple plays. Further research and development in the field will lead to new variations of existing algorithms and advancements in reinforcement learning could lead to new applications of multi-armed bandit algorithms in a world that is constantly changing.

References

[1] Pike-Burke C. *Learning Agents*; 2023. [Accessed: 26th May 2023].

[2] Slivkins A. *Introduction to Multi-Armed Bandits*; 2022. [Accessed: 17th June 2023]. Available from: https://arxiv.org/pdf/1904.07272.pdf.

[3] Duchi J. *CS229 Supplemental Lecture notes: Hoeffding's inequality*;. [Accessed: 10th June 2023]. Available from: http://cs229.stanford.edu/extra-notes/hoeffding.pdf.

[4] Lattimore T, Szepesvari C. *Bandit Algorithms*;. [Accessed: 18th June 2023]. Available from: https://tor-lattimore.com/downloads/book/book.pdf.

[5] Encyclopedia of Mathematics. *Pareto distribution*;. [Accessed: 16th June 2023]. Available from: http://encyclopediaofmath.org/index.php?title=Pareto_distribution&oldid=49651.

[6] Kun J. *Adversarial Bandits and the Exp3 Algorithm*; 2013. [Accessed: 11.06.2023]. Available from: https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/#:~:text=Exp3%20stands%20for%20Exponential-weight%20algorithm%20for%20Exploration%20and,relevant%20weights%20when%20a%20payoff%20is%20good%20%28bad%29.

[7] Jiao J. *Adversarial Bandits and EXP3*, Lecture 11; 2021. [Accessed: 13th June 2023]. Available from: https://people.eecs.berkeley.edu/~jiantao/2902021spring/scribe/EE290_Lecture_11.pdf.

[8] Abernethy J. *EXP3 Algorithm*, Lecture 20; 2013. [Accessed : 15th June 2023]. Available from: https://web.eecs.umich.edu/~jabernet/eecs598course/fall2013/web/notes/lec20_111313.pdf.

[9] Burtini G, Loeppky J, Lawrence R. *Contextual Bandits for Experiments with Covariates*. In: A Survey of Online Experiment Design with the Stochastic Multi-Armed Bandit. UBC; 2015. Available from: https://arxiv.org/pdf/1510.00757.pdf.

[10] Komiyama J, Honda J, Nakagawa H. *Optimal Regret Analysis of Thompson Sampling in Stochastic Multi-armed Bandit Problem with Multiple Plays*. In: International Conference on Machine Learning. vol. 37; 2015. Available from: http://proceedings.mlr.press/v37/komiyama15.pdf.