CAMBRIDGE
Cambridge University Press
978-1-108-48682-8 — Bandit Algorithms
Tor Lattimore , Csaba Szepesvári
Excerpt
More Information

Part I

Bandits, Probability and Concentration

1 Introduction

Bandit problems were introduced by William R. Thompson in an article published in 1933 in *Biometrika*. Thompson was interested in medical trials and the cruelty of running a trial blindly, without adapting the treatment allocations on the fly as the drug appears more or less effective. The name comes from the 1950s, when Frederick Mosteller and Robert Bush decided to study animal learning and ran trials on mice and then on humans. The mice faced the dilemma of choosing to go left or right after starting in the bottom of a T-shaped maze, not knowing each time at which end they would find food. To study a similar learning setting in humans, a 'two-armed bandit' machine was commissioned where humans could choose to pull either the left or the right arm of the machine, each giving a random pay-off with the distribution of pay-offs for each arm unknown to the human player. The machine was called a 'two-armed bandit' in homage to the one-armed bandit, an old-fashioned name for a lever-operated slot machine ('bandit' because they steal your money).

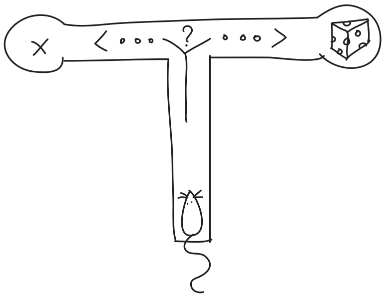

Figure 1.1 Mouse learning a T-maze.

There are many reasons to care about bandit problems. Decision-making with uncertainty is a challenge we all face, and bandits provide a simple model of this dilemma. Bandit problems also have practical applications. We already mentioned clinical trial design, which researchers have used to motivate their work for 80 years. We can't point to an example where bandits have actually been used in clinical trials, but adaptive experimental design is gaining popularity and is actively encouraged by the US Food and Drug Administration, with the justification that not doing so can lead to the withholding of effective drugs until long after a positive effect has been established.

While clinical trials are an important application for the future, there are applications where bandit algorithms are already in use. Major tech companies use bandit algorithms for configuring web interfaces, where applications include news recommendation, dynamic pricing and ad placement. A bandit algorithm plays a role in Monte Carlo Tree Search, an algorithm made famous by the recent success of AlphaGo.

Finally, the mathematical formulation of bandit problems leads to a rich structure with connections to other branches of mathematics. In writing this book (and previous papers), we have read books on convex analysis/optimisation, Brownian motion, probability theory, concentration analysis, statistics, differential geometry, information theory, Markov chains, computational complexity and more. What fun!

A combination of all these factors has led to an enormous growth in research over the last two decades. Google Scholar reports less than 1000, then 2700 and 7000 papers when searching for the phrase ‘bandit algorithm’ for the periods of 2001–5, 2006–10, and 2011–15, respectively, and the trend just seems to have strengthened since then, with 5600 papers coming up for the period of 2016 to the middle of 2018. Even if these numbers are somewhat overblown, they are indicative of a rapidly growing field. This could be a fashion, or maybe there is something interesting happening here. We think that the latter is true.

A Classical Dilemma

Imagine you are playing a two-armed bandit machine and you already pulled each lever five times, resulting in the following pay-offs (in dollars):

ROUND	1	2	3	4	5	6	7	8	9	10
LEFT	0		10	0		0				10
RIGHT		10			0		0	0	0	

Figure 1.2 Two-armed bandit

The left arm appears to be doing slightly better. The average pay-off for this arm is \$4, while the average for the right arm is only \$2. Let’s say you have 10 more trials (pulls) altogether. What is your strategy? Will you keep pulling the left arm, ignoring the right? Or would you attribute the poor performance of the right arm to bad luck and try it a few more times? How many more times? This illustrates one of the main interests in bandit problems. They capture the fundamental dilemma a learner faces when choosing between uncertain options. Should one explore an option that looks inferior or exploit by going with the option that looks best currently? Finding the right balance between exploration and exploitation is at the heart of all bandit problems.

1.1 The Language of Bandits

A bandit problem is a sequential game between a **learner** and an **environment**. The game is played over n rounds, where n is a positive natural number called the **horizon**. In each round $t \in [n]$, the learner first chooses an action A_t from a given set $\mathcal{A}$, and the environment then reveals a reward $X_t \in \mathbb{R}$.

> In the literature, actions are often also called ‘arms’. We talk about k**-armed bandits** when the number of actions is k, and about **multi-armed bandits** when the number of arms is at least two and the actual number is immaterial to the discussion. If there are multi-armed bandits, there are also **one-armed bandits**, which are really two-armed bandits where the pay-off of one of the arms is a known fixed deterministic number.

Cambridge University Press
978-1-108-48682-8 — Bandit Algorithms
Tor Lattimore , Csaba Szepesvári
Excerpt
More Information

Of course the learner cannot peek into the future when choosing their actions, which means that A_t should only depend on the **history** $H_{t-1} = (A_1, X_1, \dots, A_{t-1}, X_{t-1})$. A **policy** is a mapping from histories to actions: A learner adopts a policy to interact with an environment. An environment is a mapping from history sequences ending in actions to rewards. Both the learner and the environment may randomise their decisions, but this detail is not so important for now. The most common objective of the learner is to choose actions that lead to the largest possible cumulative reward over all n rounds, which is $\sum_{t=1}^n X_t$.

The fundamental challenge in bandit problems is that the environment is unknown to the learner. All the learner knows is that the true environment lies in some set $\mathcal{E}$ called the **environment class**. Most of this book is about designing policies for different kinds of environment classes, though in some cases the framework is extended to include side observations as well as actions and rewards.

The next question is how to evaluate a learner. We discuss several performance measures throughout the book, but most of our efforts are devoted to understanding the **regret**. There are several ways to define this quantity. To avoid getting bogged down in details, we start with a somewhat informal definition.

Definition 1.1. The regret of the learner relative to a policy π (not necessarily that followed by the learner) is the difference between the total expected reward using policy π for n rounds and the total expected reward collected by the learner over n rounds. The regret relative to a set of policies Π is the maximum regret relative to any policy $\pi \in \Pi$ in the set.

The set Π is often called the **competitor class**. Another way of saying all this is that the regret measures the performance of the learner relative to the best policy in the competitor class. We usually measure the regret relative to a set of policies Π that is large enough to include the optimal policy for all environments in $\mathcal{E}$. In this case, the regret measures the loss suffered by the learner relative to the optimal policy.

Example 1.2. Suppose the action set is $\mathcal{A} = \{1, 2, \dots, k\}$. An environment is called a **stochastic Bernoulli bandit** if the reward $X_t \in \{0, 1\}$ is binary valued and there exists a vector $\mu \in [0, 1]^k$ such that the probability that $X_t = 1$ given the learner chose action $A_t = a$ is μ_a. The class of stochastic Bernoulli bandits is the set of all such bandits, which are characterised by their mean vectors. If you knew the mean vector associated with the environment, then the optimal policy is to play the fixed action $a^* = \operatorname{argmax}_{a \in \mathcal{A}} \mu_a$. This means that for this problem the natural competitor class is the set of k constant polices $\Pi = \{\pi_1, \dots, \pi_k\}$, where π_i chooses action i in every round. The regret over n rounds becomes

$$R_n = n \max_{a \in \mathcal{A}} \mu_a - \mathbb{E}\left[\sum_{t=1}^n X_t\right],$$

where the expectation is with respect to the randomness in the environment and policy. The first term in this expression is the maximum expected reward using any policy. The second term is the expected reward collected by the learner.

For a fixed policy and competitor class, the regret depends on the environment. The environments where the regret is large are those where the learner is behaving worse. Of course the ideal case is that the regret be small for all environments. The **worst-case regret** is the maximum regret over all possible environments.

One of the core questions in the study of bandits is to understand the growth rate of the regret as n grows. A good learner achieves sublinear regret. Letting R_n denote the regret over n rounds, this means that $R_n = o(n)$ or equivalently that $\lim_{n\to\infty} R_n/n = 0$. Of course one can ask for more. Under what circumstances is $R_n = O(\sqrt{n})$ or $R_n = O(\log(n))$? And what are the leading constants? How does the regret depend on the specific environment in which the learner finds itself? We will discover eventually that for the environment class in Example 1.2, the worst-case regret for any policy is at least $\Omega(\sqrt{n})$ and that there exist policies for which $R_n = O(\sqrt{n})$.

A large environment class corresponds to less knowledge by the learner. A large competitor class means the regret is a more demanding criteria. Some care is sometimes required to choose these sets appropriately so that *(a)* guarantees on the regret are meaningful and *(b)* there exist policies that make the regret small.

The framework is general enough to model almost anything by using a rich enough environment class. This cannot be bad, but with too much generality it becomes impossible to say much. For this reason, we usually restrict our attention to certain kinds of environment classes and competitor classes.

A simple problem setting is that of **stochastic stationary bandits**. In this case the environment is restricted to generate the reward in response to each action from a distribution that is specific to that action and independent of the previous action choices and rewards. The environment class in Example 1.2 satisfies these conditions, but there are many alternatives. For example, the rewards could follow a Gaussian distribution rather than Bernoulli. This relatively mild difference does not change the nature of the problem in a significant way. A more drastic change is to assume the action set $\mathcal{A}$ is a subset of $\mathbb{R}^d$ and that the mean reward for choosing some action $a \in \mathcal{A}$ follows a linear model, $X_t = \langle a, \theta \rangle + \eta_t$ for $\theta \in \mathbb{R}^d$ and η_t a standard Gaussian (zero mean, unit variance). The unknown quantity in this case is θ, and the environment class corresponds to its possible values ($\mathcal{E} = \mathbb{R}^d$).

For some applications, the assumption that the rewards are stochastic and stationary may be too restrictive. The world mostly appears deterministic, even if it is hard to predict and often chaotic looking. Of course, stochasticity has been enormously successful in explaining patterns in data, and this may be sufficient reason to keep it as the modelling assumption. But what if the stochastic assumptions fail to hold? What if they are violated for a single round? Or just for one action, at some rounds? Will our best algorithms suddenly perform poorly? Or will the algorithms developed be robust to smaller or larger deviations from the modelling assumptions?

An extreme idea is to drop all assumptions on how the rewards are generated, except that they are chosen without knowledge of the learner's actions and lie in a bounded set. If these are the only assumptions, we get what is called the setting of **adversarial bandits**. The trick to say something meaningful in this setting is to restrict the competitor class. The learner is not expected to find the best sequence of actions, which may be like finding a needle in a haystack. Instead, we usually choose Π to be the set of constant policies and demand that the learner is not much worse than any of these. By defining the regret in this way, the stationarity assumption is transported into the definition of regret rather than constraining the environment.

Of course there are all shades of grey between these two extremes. Sometimes we consider the case where the rewards are stochastic, but not stationary. Or one may analyse the robustness of an algorithm for stochastic bandits to small adversarial perturbations. Another idea is to isolate exactly which properties of the stochastic assumption are really exploited by a policy designed for stochastic bandits. This kind of inverse analysis can help explain the strong performance of policies when facing environments that clearly violate the assumptions they were designed for.

1.1.1 Other Learning Objectives

We already mentioned that the regret can be defined in several ways, each capturing slightly different aspects of the behaviour of a policy. Because the regret depends on the environment, it becomes a multi-objective criterion: ideally, we want to keep the regret small across all possible environments. One way to convert a multi-objective criterion into a single number is to take averages. This corresponds to the Bayesian viewpoint where the objective is to minimise the average cumulative regret with respect to a prior on the environment class.

Maximising the sum of rewards is not always the objective. Sometimes the learner just wants to find a near-optimal policy after n rounds, but the actual rewards accumulated over those rounds are unimportant. We will see examples of this shortly.

1.1.2 Limitations of the Bandit Framework

One of the distinguishing features of all bandit problems studied in this book is that the learner never needs to plan for the future. More precisely, we will invariably make the assumption that the learner's available choices and rewards tomorrow are not affected by their decisions today. Problems that do require this kind of long-term planning fall into the realm of **reinforcement learning**, which is the topic of the final chapter. Another limitation of the bandit framework is the assumption that the learner observes the reward in every round. The setting where the reward is not observed is called **partial monitoring** and is the topic of Chapter 37. Finally, often, the environment itself consists of strategic agents, which the learner needs to take into account. This problem is studied in game theory and would need a book on its own.

1.2 Applications

After this short preview, and as an appetiser before the hard work, we briefly describe the formalisations of a variety of applications.

A/B Testing

The designers of a company website are trying to decide whether the 'buy it now' button should be placed at the top of the product page or at the bottom. In the old days, they would

Cambridge University Press
978-1-108-48682-8 — Bandit Algorithms
Tor Lattimore , Csaba Szepesvári
Excerpt
More Information

commit to a trial of each version by splitting incoming users into two groups of 10 000. Each group would be shown a different version of the site, and a statistician would examine the data at the end to decide which version was better. One problem with this approach is the non-adaptivity of the test. For example, if the effect size is large, then the trial could be stopped early.

One way to apply bandits to this problem is to view the two versions of the site as actions. Each time t a user makes a request, a bandit algorithm is used to choose an action $A_t \in \mathcal{A} = \{\textsc{SiteA}, \textsc{SiteB}\}$, and the reward is $X_t = 1$ if the user purchases the product and $X_t = 0$ otherwise.

In traditional A/B testing, the objective of the statistician is to decide which website is better. When using a bandit algorithm, there is no need to end the trial. The algorithm automatically decides when one version of the site should be shown more often than another. Even if the real objective is to identify the best site, then adaptivity or early stopping can be added to the A/B process using techniques from bandit theory. While this is not the focus of this book, some of the basic ideas are explained in Chapter 33.

Advert Placement

In advert placement, each round corresponds to a user visiting a website, and the set of actions $\mathcal{A}$ is the set of all available adverts. One could treat this as a standard multi-armed bandit problem, where in each round a policy chooses $A_t \in \mathcal{A}$, and the reward is $X_t = 1$ if the user clicked on the advert and $X_t = 0$ otherwise. This might work for specialised websites where the adverts are all likely to be appropriate. But for a company like Amazon, the advertising should be targeted. A user that recently purchased rock-climbing shoes is much more likely to buy a harness than another user. Clearly an algorithm should take this into account.

The standard way to incorporate this additional knowledge is to use the information about the user as **context**. In its simplest formulation, this might mean clustering users and implementing a separate bandit algorithm for each cluster. Much of this book is devoted to the question of how to use side information to improve the performance of a learner.

This is a good place to emphasise that the world is messy. The set of available adverts is changing from round to round. The feedback from the user can be delayed for many rounds. Finally, the real objective is rarely just to maximise clicks. Other metrics such as user satisfaction, diversity, freshness and fairness, just to mention a few, are important too. These are the kinds of issues that make implementing bandit algorithms in the real world a challenge. This book will not address all these issues in detail. Instead we focus on the foundations and hope this provides enough understanding that you can invent solutions for whatever peculiar challenges arise in your problem.

Recommendation Services

Netflix has to decide which movies to place most prominently in your 'Browse' page. Like in advert placement, users arrive at the page sequentially, and the reward can be measured as some function of (a) whether or not you watched a movie and (b) whether or not you rated it positively. There are many challenges. First of all, Netflix shows a long list of movies, so the set of possible actions is combinatorially large. Second, each user watches relatively few movies, and individual users are different. This suggests approaches such as low-rank matrix factorisation (a popular approach in 'collaborative filtering'). But notice this is not an offline problem. The learning algorithm gets to choose what users see and this affects the data. If the users are never recommended the AlphaGo movie, then few users will watch it, and the amount of data about this film will be scarce.

Network Routing

Another problem with an interesting structure is network routing, where the learner tries to direct internet traffic through the shortest path on a network. In each round the learner receives the start/end destinations for a packet of data. The set of actions is the set of all paths starting and ending at the appropriate points on some known graph. The feedback in this case is the time it takes for the packet to be received at its destination, and the reward is the negation of this value. Again the action set is combinatorially large. Even relatively small graphs have an enormous number of paths. The routing problem can obviously be applied to more physical networks such as transportation systems used in operations research.

Dynamic Pricing

In dynamic pricing, a company is trying to automatically optimise the price of some product. Users arrive sequentially, and the learner sets the price. The user will only purchase the product if the price is lower than their valuation. What makes this problem interesting is (a) the learner never actually observes the valuation of the product, only the binary signal that the price was too low/too high, and (b) there is a monotonicity structure in the pricing. If a user purchased an item priced at \$10, then they would surely purchase it for \$5, but whether or not it would sell when priced at \$11 is uncertain. Also, the set of possible actions is close to continuous.

Waiting Problems

Every day you travel to work, either by bus or by walking. Once you get on the bus, the trip only takes 5 minutes, but the timetable is unreliable, and the bus arrival time is unknown and stochastic. Sometimes the bus doesn't come at all. Walking, on the other hand, takes 30 minutes along a beautiful river away from the road. The problem is to devise a policy for choosing how long to wait at the bus stop before giving up and walking to minimise the time to get to your workplace. Walk too soon, and you miss the bus and gain little information. But waiting too long also comes at a price.

While waiting for a bus is not a problem we all face, there are other applications of this setting. For example, deciding the amount of inactivity required before putting a hard drive into sleep mode or powering off a car engine at traffic lights. The statistical part of the waiting problem concerns estimating the cumulative distribution function of the bus arrival times from data. The twist is that the data is censored on the days you chose to walk before the bus arrived, which is a problem analysed in the subfield of statistics called survival analysis. The interplay between the statistical estimation problem and the challenge of balancing exploration and exploitation is what makes this and the other problems studied in this book interesting.

Resource Allocation

A large part of operations research is focussed on designing strategies for allocating scarce resources. When the dynamics of demand or supply are uncertain, the problem has elements reminiscent of a bandit problem. Allocating too few resources reveals only partial information about the true demand, but allocating too many resources is wasteful. Of course, resource allocation is broad, and many problems exhibit structure that is not typical of bandit problems, like the need for long-term planning.

Tree Search

The UCT algorithm is a tree search algorithm commonly used in perfect-information game-playing algorithms. The idea is to iteratively build a search tree where in each iteration the algorithm takes three steps: *(1)* chooses a path from the root to a leaf; *(2)* expands the leaf (if possible); *(3)* performs a Monte Carlo roll-out to the end of the game. The contribution of a bandit algorithm is in selecting the path from the root to the leaves. At each node in the tree, a bandit algorithm is used to select the child based on the series of rewards observed through that node so far. The resulting algorithm can be analysed theoretically, but more importantly has demonstrated outstanding empirical performance in game-playing problems.

1.3 Notes

1 The reader may find it odd that at one point we identified environments with maps from histories to rewards, while we used the language that a learner 'adopts a policy' (a map from histories to actions). The reason is part historical and part because policies and their design are at the center of the book, while the environment strategies will mostly be kept fixed (and relatively simple). On this note, strategy is also a word that sometimes used interchangeably with policy.

1.4 Bibliographic Remarks

As we mentioned in the very beginning, the first paper on bandits was by Thompson [1933]. The experimentation on mice and humans that led to the name comes from the paper by Bush and Mosteller [1953]. Much credit for the popularisation of the field must go to famous mathematician and statistician, Herbert Robbins, whose name appears on many of the works that we reference, with the earliest being: [Robbins, 1952]. Another early pioneer is Herman Chernoff, who wrote papers with titles like 'Sequential Decisions in the Control of a Spaceship' [Bather and Chernoff, 1967].

Besides these seminal papers, there are already a number of books on bandits that may serve as useful additional reading. The most recent (and also most related) is by Bubeck and Cesa-Bianchi [2012] and is freely available online. This is an excellent book and is warmly recommended. The main difference between their book and ours is that *(a)* we have the benefit of seven years of additional research in a fast-moving field and *(b)* our longer page limit permits more depth. Another relatively recent book is *Prediction, Learning and Games* by Cesa-Bianchi and Lugosi [2006]. This is a wonderful book, and quite comprehensive. But its scope is 'all of' online learning, which is so broad that bandits are not covered in great depth. We should mention there is also a recent book on bandits by Slivkins [2019]. Conveniently it covers some topics not covered in this book (notably Lipschitz bandits and bandits with knapsacks). The reverse is also true, which should not be surprising since our

