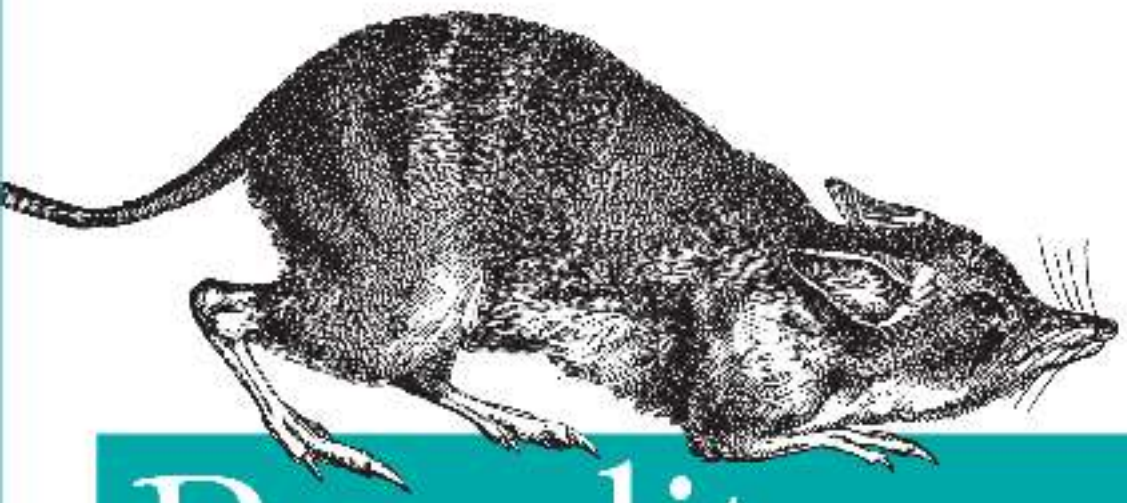
Developing, Deploying, and Debugging



Bandit Algorithms

for Website Optimization

O'REILLY®

John Myles White

Bandit Algorithms for Website Optimization

John Myles White

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Bandit Algorithms for Website Optimization

by John Myles White

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Mike Loukides and Meghan Blanchette
Production Editor: Christopher Hearse
Proofreader: Christopher Hearse
Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

December 2012: First Edition

Revision History for the First Edition:

2012-12-07 First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449341336* for release details.

ISBN: 978-1-449-34133-6

LSI

Table of Contents

Preface

Finding the Code for This Book

This book is about algorithms. But it's not a book about the theory of algorithms. It's a short tutorial introduction to algorithms that's targetted at people who like to learn about new ideas by experimenting with them in practice.

Because we want you to experiment, this book is meant to be read while you're near an interpreter for your favorite programming language. In the text, we illustrate every algorithm we describe using Python. As part of the accompanying online materials, there is similar code available that implements all of the same algorithms in Julia, a new programming language that is ideally suited for implementing bandit algorithms. Alongside the Python and Julia code, there are also links to similar implementations in other languages like JavaScript.

We've chosen to use Python for this book because it seems like a reasonable lingua franca for programmers. If Python isn't your style, you should be able to translate our Python code into your favorite programming language fairly easily.

Assuming you are happy using Python or Julia, you can find the code for the book on GitHub at *https://github.com/johnmyleswhite/BanditsBook*. If you find mistakes or would like to submit an implementation in another language, please make a pull request.

Dealing with Jargon: A Glossary

While this book isn't meant to introduce you to the theoretical study of the Multiarmed Bandit Problem or to prepare you to develop novel algorithms for solving the problem, we want you to leave this book with enough understanding of existing work to be able to follow the literature on the Multiarmed Bandit Problem. In order to do that, we have to introduce quite a large number of jargon words. These jargon words can be a little

odd at a first, but they're universally used in the academic literature on Multiarmed Bandit Problems. As you read this book, you will want to return to the list of jargon words below to remind yourself what they mean. For now, you can glance through them, but we don't expect you to understand these words yet. Just know that this material is here for you to refer back to if you're ever confused about a term we use.

Reward
: A quantitative measure of success. In business, the ultimate rewards are profits, but we can often treat simpler metrics like click-through rates for ads or sign-up rates for new users as rewards. What matters is that (A) there is a clear quantitative scale and (B) it's better to have more reward than less reward.

Arm
: What options are available to us? What actions can we take? In this book, we'll refer to the options available to us as arms. The reasons for this naming convention will be easier to understand after we've discuss a little bit of the history of the Multiarmed Bandit Problem.

Bandit
: A bandit is a collection of arms. When you have many options available to you, we call that collection of options a Multiarmed Bandit. A Multiarmed Bandit is a mathematical model you can use to reason about how to make decisions when you have many actions you can take and imperfect information about the rewards you would receive after taking those actions. The algorithms presented in this book are ways of trying to solve the problem of deciding which arms to pull when. We refer to the problem of choosing arms to pull as the Multiarmed Bandit Problem.

Play/Trial
: When you deal with a bandit, it's assumed that you get to pull on each arm multiple times. Each chance you have to pull on an arm will be called a play or, more often, a trial. The term "play" helps to invoke the notion of gambling that inspires the term "arm", but the term trial is quite commonly used.

Horizon
: How many trials will you have before the game is finished? The number of trials left is called the horizon. If the horizon is short, you will often use a different strategy than you would use if the horizon were long, because having many chances to play each arm means that you can take greater risks while still having time to recover if anything goes wrong.

Exploitation
: An algorithm for solving the Multiarmed Bandit Problem *exploits* if it plays the arm with the highest estimated value based on previous plays.

Exploration
: An algorithm for solving the Multiarmed Bandit Problem *explores* if it plays any arm that does not have the highest estimated value based on previous plays. In other words, exploration occurs whenever exploitation does not.

Explore/Exploit Dilemma
: The observation that any learning system must strike a compromise between its impulse to explore and its impulse to exploit. The dilemma has no exact solution, but the algorithms described in this book provide useful strategies for resolving the conflicting goals of exploration and exploitation.

Annealing
: An algorithm for solving the Multiarmed Bandit Problem anneals if it explores less over time.

Temperature
: A parameter that can be adjusted to increase the amount of exploration in the Softmax algorithm for solving the Multiarmed Bandit Problem. If you decrease the temperature parameter over time, this causes the algorithm to anneal.

Streaming Algorithms
: An algorithm is a streaming algorithm if it can process data one piece at a time. This is the opposite of batch processing algorithms that need access to all of the data in order to do anything with it.

Online Learning
: An algorithm is an online learning algorithm if it can not only process data one piece at a time, but can also provide provisional results of its analysis after each piece of data is seen.

Active Learning
: An algorithm is an active learning algorithm if it can decide which pieces of data it wants to see next in order to learn most effectively. Most traditional machine learning algorithm are not active: they passively accept the data we feed them and do not tell us what data we should collect next.

Bernoulli
: A Bernoulli system outputs a `1` with probability `p` and a `0` with probability `1 - p`.

Conventions Used in This Book

The following typographical conventions are used in this book:

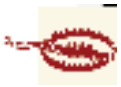
Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if this book includes code examples, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'™Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product'™s documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Bandit Algorithms for Website Optimization* by John Myles White. Copyright 2013 John Myles White, 978-1-449-34133-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://oreilly.com/mult_algorithms_optimization*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

This minibook has benefitted from years of discussions about the Explore-Exploit problem that I've had with the members of Princeton's psychology department. My thanks go to them, as well as to the three technical reviewers—Matt Gershoff at Conductrics, Roberto Medri at Esty, and Tim Hopper at RTI—all of whom read through this book and found countless ways to improve it. Their comments were invaluable and I'm deeply appreciative for all of the little errors that they kept from creeping into the final release of this book. Finally, I'd like to thank the various people who've contributed to the codebase on bandit algorithms that complements this book. Receiving pull requests contributing supplemental code for a book that wasn't even released has been among my favorite experiences ever as an author.

CHAPTER 1

Two Characters: Exploration and Exploitation

To set the stage for this book, I'm going to tell you a short story about a web developer, Deborah Knull, who ran a small web business that provided most of her income. Deb Knull's story will introduce the core concepts that come up when studying bandit algorithms, which are called *exploration* and *exploitation*. To make those ideas concrete, I'm going to associate them with two types of people: a scientist who explores and a businessman who exploits. My hope is that these two characters will help you understand why you need to find a way to balance the desires of both of these types of people in order to build a better website.

The Scientist and the Businessman

One Sunday morning, a young web enterpreneur, Deb Knull, came to suspect that changing the primary color of her site's logo would make her site's users feel more comfortable. Perhaps more importantly, she thought that making her customers feel more comfortable would make them buy more of the products her site was selling.

But Deb Knull worried that a new color could potentially disorient users and make them feel less comfortable. If that were true, her clever idea to increase sales might actually make her users buy fewer products instead. Unsure which of her instincts to trust, she asked for advice from two of her friends: Cynthia, a scientist, and Bob, a businessman.

Cynthia the Scientist

Cynthia, the scientist, loved Deb's proposed logo change. Excited by the opportunity to try out someting new, Cynthia started to lecture Deb about how to test her change carefully: "You can't just switch your logo to a new color and then assume that the change

in the logo's color is responsible for whatever happens next. You'll need to run a controlled experiment. If you don't test your idea with a controlled experiment, you'll never know whether the color change actually helped or hurt your sales. After all, it's going to be Christmas season soon. If you change the logo now, I'm sure you'll see a huge increase in sales relative to the last two months. But that's not informative about the merits of the new logo: for all you know, the new color for your logo might actually be hurting sales."

"Christmas is such a lucrative time of year that you'll see increased profits despite having made a bad decision by switching to a new color logo. If you want to know what the real merit of your idea is, you need to make a proper apples-to-apples comparison. And the only way I know how to do that is to run a traditional randomized experiment: whenever a new visitor comes to your site, you should flip a coin. If it comes up heads, you'll put that new visitor into Group A and show them the old logo. If it comes up tails, you'll put the visitor into Group B and show them the new logo. Because the logo you show each user is selected completely randomly, any factors that might distort the comparison between the old logo and new logo should balance out over time. If you use a coinflip to decide which logo to show each user, the effect of the logo won't be distorted by the effects of other things like the Christmas season."

Deb agreed that she shouldn't just switch the color of her logo over; as Cynthia the scientist was suggesting, Deb saw that she needed to run a controlled experiment to assess the business value of changing her site's logo.

In Cynthia's proposed A/B testing setup, Groups A and B of users would see slightly different versions of the same website. After enough users had been exposed to both designs, comparisons between the two groups would allow Deb to decide whether the proposed change would help or hurt her site.

Once she was convinced of the merits of A/B testing, Deb started to contemplate much larger scale experiments: instead of running an A/B test, she started to consider comparing her old black logo with six other colors, including some fairly quirky colors like purple and chartreuse. She'd gone from A/B testing to A/B/C/D/E/F/G testing in a matter of minutes.

Running careful experiments about each of these ideas excited Cynthia as a scientist, but Deb worried that some of the colors that Cynthia had proposed testing seemed likely to be much worse than her current logo. Unsure what to do, Deb raised her concerns with Bob, who worked at a large multinational bank.

Bob the Businessman

Bob heard Deb's idea of testing out several new logo colors on her site and agreed that experimentation could be profitable. But Bob was also very skeptical about the value of trying out some of the quirkier of Cynthia's ideas.

"Cynthia's a scientist. Of course she thinks that you should run lots of experiments. She wants to have knowledge for knowledge's sake and never thinks about the costs of her experiments. But you're a businesswoman, Deb. You have a livelihood to make. You should try to maximize your site's profits. To keep your checkbook safe, you should only run experiments that could be profitable. Knowledge is only valuable for profit's sake in business. Unless you really believe a change has the potential to be valuable, don't try it at all. And if you don't have any new ideas that you have faith in, going with your traditional logo is the best strategy."

Bob's skepticism of the value of large-scale experimentation rekindled Deb's concerns earlier: the threat of losing customers was greater than Deb had felt when energized by Cynthia's passion for designing experiments. But Deb also wasn't clear how to decide which changes would be profitable without trying them out, which seemed to lead her back to Cynthia's original proposal and away from Bob's preference for tradition.

After spending some time weighing Cynthia and Bob's arguments, Deb decided that there was always going to be a fundamental trade-off between the goals that motivated Cynthia and Bob: a small business couldn't afford to behave like a scientist and spend money gaining knowledge for knowledge's sake, but it also couldn't afford to focus shortsightedly on current profits and to never try out any new ideas. As far as she could see, Deb felt that there was never going to be a simple way to balance the need to (1) learn new things and (2) profit from old things that she'd already learned.

Oscar the Operations Researcher

Luckily, Deb had one more friend she knew she could turn to for advice: Oscar, a professor who worked in the local Department of Operations Research. Deb knew that Oscar was an established expert in business decision-making, so she suspected the Oscar would have something intelligent to say about her newfound questions about balancing experimentation with profit-maximization.

And Oscar was indeed interested in Deb's idea:

"I entirely agree that you have to find a way to balance Cynthia's interest in experimentation and Bob's interest in profits. My colleagues and I call that the Explore-Exploit trade-off."

"Which is?"

"It's the way Operations Researchers talk about your need to balance experimentation with profit-maximization. We call experimentation *exploration* and we call profit-maximization *exploitation*. They're the fundamental values that any profit-seeking system, whether it's a person, a company or a robot, has to find a way to balance. If you do too much exploration, you lose money. And if you do too much exploitation, you stagnate and miss out on new opportunities."

"So how do I balance exploration and exploitation?"

"Unfortunately, I don't have a simple answer for you. Like you suspected, there is no universal solution to balancing your two goals: to learn which ideas are good or bad, you have to explore — at the risk of losing money and bringing in fewer profits. The right way to choose between exploring new ideas and exploiting the best of your old ideas depends on the details of your situation. What I can tell you is that your plan to run A/B testing, which both Cynthia and Bob seem to be taking for granted as the only possible way you could learn which color logo is best, is not always the best option."

"For example, a trial period of A/B testing followed by sticking strictly to the *best* design afterwards only makes sense if there is a definite best design that consistently works across the Christmas season and the rest of the year. But imagine that the best color scheme is black/orange near Halloween and red/green near Christmas. If you run an A/B experiment during only one of those two periods of time, you'll come to think there's a huge difference — and then your profits will suddenly come crashing down during the other time of year."

"And there are other potential problems as well with naive A/B testing: if you run an experiment that streches across both times of year, you'll see no average effect for your two color schemes — even though there's a huge effect in each of the seasons if you had examined them separately. You need context to design meaningful experiments. And you need to experiment intelligently. Thankfully, there are lots of algorithms you can use to help you design better experiments."

The Explore-Exploit Dilemma

Hopefully the short story I've just told you has made it clear to you that you have two completely different goals you need to address when you try to optimize a website: you need to (A) learn about new ideas (which we'll always call *exploring* from now on), while you also need to (B) take advantage of the best of your old ideas (which we'll always call *exploiting* from now on). Cynthia the scientist was meant to embody exploration: she was open to every new idea, including the terrible ideas of using a purple or chartreuse logo. Bob was meant to embody exploitation, because he closes his mind to new ideas prematurely and is overly willing to stick with tradition.

To help you build better websites, we'll do exactly what Oscar would have done to help Deborah: we'll give you a crash course in methods for solving the Explore-Exploit dilemma. We'll discuss two classic algorithms, one state-of-the-art algorithm and then refer you to standard textbooks with much more information about the huge field that's arisen around the Exploration-Exploitation trade-off.

But, before we start working with algorithms for solving the Exploration-Exploitation trade-off, we're going to focus on the differences between the bandit algorithms we'll present in this book and the tradition A/B testing methods that most web developers would use to explore new ideas.

CHAPTER 2

Why Use Multiarmed Bandit Algorithms?

What Are We Trying to Do?

In the previous chapter, we introduced the two core concepts of exploration and exploitation. In this chapter, we want to make those concepts more concrete by explaining how they would arise in the specific context of website optimization. When we talk about "optimizing a website", we're referring to a step-by-step process in which a web developer makes a series of changes to a website, each of which is meant to increase the success of that site. For many web developers, the most famous type of website optimization is called Search Engine Optimization (or SEO for short), a process that involves modifying a website to increase that site's rank in search engine results. We won't discuss SEO at all in this book, but the algorithms that we will describe can be easily applied as part of an SEO campaign in order to decide which SEO techniques work best.

Instead of focusing on SEO, or on any other sort of specific modification you could make to a website to increase its success, we'll be describing a series of algorithms that allow you to measure the real-world value of any modifications you might make to your site(s).

But, before we can describe those algorithms, we need to make sure that we all mean the same thing when we use the word "success." From now on, we are only going to use the word "success" to describe measurable achievements like:

Traffic
: Did a change increase the amount of traffic to a site's landing page?

Conversions
: Did a change increase the number of one-time vistors who were successfully converted into repeat customers?

Sales
: Did a change increase the number of purchases being made on a site by either new or existing customers?

CTR's
: Did a change increase the number of times that visitors clicked on an ad?

In addition to an unambiguous, quantitative measurement of success, we're going to also need to have a list of potential changes you believe might increase the success of your site(s). From now on, we're going to start calling our measure of success a *reward* and our list of potential changes *arms*. The historical reasons for those terms will be described shortly. We don't personally think they're very well-chosen terms, but they're absolutely standard in the academic literature on this topic and will help us make our discussion of algorithms precise.

For now, we want to focus on a different issue: why should we even bother using bandit algorithms to test out new ideas when optimizing websites? Isn't A/B testing already sufficient?

To answer those questions, let's describe the typical A/B testing setup in some detail and then articulate a list of reasons why it may not be ideal.

The Business Scientist: Web-Scale A/B Testing

Most large websites already know a great deal about how to test out new ideas: as described in our short story about Deb Knull, they understand that you can only determine whether a new idea works by performing a controlled experiment.

This style of controlled experimentation is called A/B testing because it typically involves randomly assigning an incoming web user to one of two groups: Group A or Group B. This random assignment of users to groups continues on for a while until the web developer becomes convinced that either Option A is more successful than Option B or, vice versa, that Option B is more successful than Option A. After that, the web developer assigns all future users to the more successful version of the website and closes out the inferior version of the website.

This experimental approach to trying out new ideas has been extremely successful in the past and will continue to be successful in many contexts. So why should we believe that the bandit algorithms described in the rest of this book have anything to offer us?

Answering this question properly requires that we return to the concepts of exploration and exploitation. Standard A/B testing consists of:

- A short period of *pure exploration*, in which you assign equal numbers of users to Groups A and B.

- A long period of *pure exploitation*, in which you send all of your users to the more successful version of your site and never come back to the option that seemed to be inferior.

Why might this be a bad strategy?

- It jumps discretely from exploration into exploitation, when you might be able to smoothly transition between the two.
- During the purely exploratory phase, it wastes resources exploring inferior options in order to gather as much data as possible. But you shouldn't want to gather data about strikingly inferior options.

Bandit algorithms provide solutions to both of these problems: (1) they smoothly decrease the amount of exploring they do over time instead of requiring you to make a sudden jump and (2) they focus your resources during exploration on the better options instead of wasting time on the inferior options that are over-explored during typical A/B testing. In fact, bandit algorithms address both of those concerns is the same way because they gradually fixate on the best available options over time. In the academic literature, this process of settling down on the best available option is called convergence. All good bandit algorithms will eventually converge.

In practice, how important those two types of improvements will be to your business depends a lot on the details of how your business works. But the general framework for thinking about exploration and exploitation provided by bandit algorithms will be useful to you no matter what you end up doing because bandit algorithms subsume A/B testing as a special case. Standard A/B testing describes one extreme case in which you jump from pure exploration to pure exploitation. Bandit algorithms let you operate in the much larger and more interesting space between those two extreme states.

In order to see how bandit algorithms achieve that balance, let's start working with our first algorithm: the epsilon-Greedy algorithm.

CHAPTER 3

The epsilon-Greedy Algorithm

Introducing the epsilon-Greedy Algorithm

To get you started thinking algorithmically about the Explore-Exploit dilemma, we're going to teach you how to code up one of the simplest possible algorithms for trading off exploration and exploitation. This algorithm is called the epsilon-Greedy algorithm. In computer science, a greedy algorithm is an algorithm that always takes whatever action seems best at the present moment, even when that decision might lead to bad long term consequences. The epsilon-Greedy algorithm is almost a greedy algorithm because it generally exploits the best available option, but every once in a while the epsilon-Greedy algorithm explores the other available options. As we'll see, the term epsilon in the algorithm's name refers to the odds that the algorithm explores instead of exploiting.

Let's be more specific. The epsilon-Greedy algorithm works by randomly oscillating between Cynthia's vision of purely randomized experimentation and Bob's instinct to maximize profits. The epsilon-Greedy algorithm is one of the easiest bandit algorithms to understand because it tries to be fair to the two opposite goals of exploration and exploitation by using a mechanism that even a little kid could understand: it just flips a coin. While there are a few details we'll have to iron out to make that statement precise, the big idea behind the epsilon-Greedy algorithm really is that simple: if you flip a coin and it comes up heads, you should explore for a moment. But if the coin comes up tails, you should exploit.

Let's flesh that idea out by continuing on with our example of changing the color of a website's logo to increase revenue. We'll assume that Deb is debating between two colors, green and red, and that she wants to find the one color that maximizes the odds that a

new visitor to her site will be converted into a registered user. The epsilon-Greedy algorithm attempts to find the best color logo using the following procedure (shown diagrammatically in Figure 3-1), which is applied to each new potential customer sequentially:

- When a new visitor comes to the site, the algorithm flips a coin that comes up tails with probability epsilon. (If you're not used to thinking in terms of probabilities, the phrase "with probability X" means that something happens 100 * X percent of the time. So saying that a coin comes up tails with probability 0.01 means that it comes up tails 1% of the time.)
- If the coin comes up heads, the algorithm is going to exploit. To exploit, the algorithm looks up the historical conversion rates for both the green and red logos in whatever data source it uses to keep track of things. After determining which color had the highest success rate in the past, the algorithm decides to show the new visitor the color that's been most successful historically.
- If, instead of coming up heads, the coin comes up tails, the algorithm is going to explore. Since exploration involves randomly experimenting with the two colors being considered, the algorithm needs to flip a second coin to choose between them. Unlike the first coin, we'll assume that this second coin comes up head 50% of the time. Once the second coin is flipped, the algorithm can move on with the last step of the procedure:
  - If the second coin comes up heads, show the new visitor the green logo.
  - If the second coin comes up tails, show the new visitor the red logo.

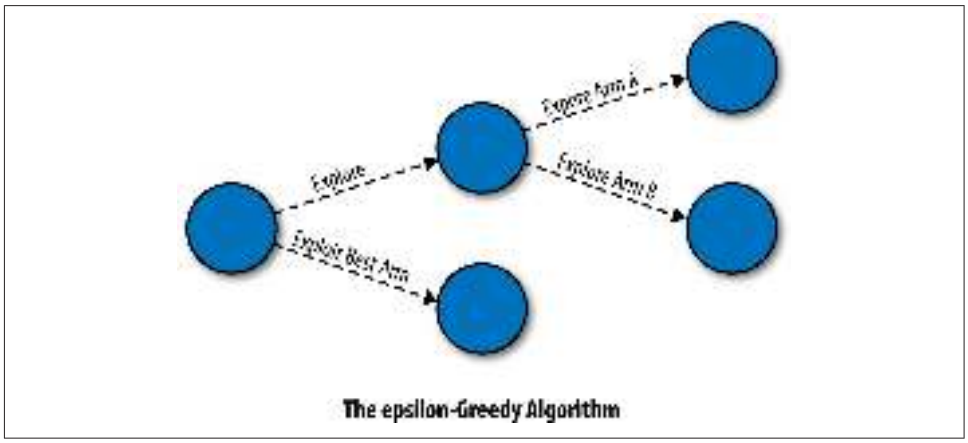


Figure 3-1. The epsilon-Greedy arm selection process