ALGORITHMS TO LIVE BY: The Computer Science of Human Decisions

by: Brian Christian & Tom Griffiths

Overview

Life is full of decisions. Where should you go on vacation, and how should you order your stops? How long should you look for an apartment before deciding? Thankfully, computer science can illuminate and rationalize decisions for us. Whether you're on a time crunch or simply stuck, you can apply algorithms to your life for optimal decision-making.

"Up against such hard cases, effective algorithms make assumptions, show a bias toward simpler solutions, trade off the costs of error against the costs of delay, and take chances."

Chapter 1: Optimal Stopping

Finding "the perfect one" can seem daunting. Take apartments: when do you stop looking and just choose? You might decide too early, leaving a better apartment unfound. Conversely, search too long and the only apartment left may not meet your standards.

This is the **Optimal Stopping Problem**: finding the right moment to stop looking. To solve it, use the **Look-Then-Leap Rule:** establish a set amount of time to look for what you need. Next spend 37% of your search time (the **37% Rule**) looking without committing. Finally, pick the first option that's better than anything you saw during the search phase.

The most famous example of the 37% Rule concerns the "secretary problem." To maximize the chance of hiring the best possible secretary, you would interview 37% of the applicants. From there, either follow the **Look-Then-Leap Rule** and hire the next best applicant or follow the **Threshold Rule** and pick the best of the bunch so far.

"Whether it involves secretaries, fiancé(e)s or apartments, life is full of optimal stopping."

Whether selling a house or parking a car, an algorithm can calculate exactly when you should stop. You can't accept the first offer because a better one may come soon after, but wait too long and the cost may outweigh your expected gain. Optimal stopping boils down to maximizing search time for maximum results. Essentially, knowing when to stop.

Chapter 2: Explore/Exploit

When deciding on a restaurant, one must consider the **Explore/Exploit Tradeoff:** exploring *new* information versus exploiting *known* information. Some might argue for the **Win-Stay, Lose-Shift** principle – sticking with a winning choice until a slip-up occurs. But what if a restaurant you disliked comes under new management? Your next experience could be drastically better! This is called the **Multi-Armed Bandit Problem**: scenarios where the only way to know the odds is to explore them first. There have been many approaches to this, although few as memorable as the **Gittins Index.** The Gittins Index favors exploration, as long as what you've learned can be exploited later.

"Exploration in itself has value, since trying new things increases our chances of finding the best. So taking the future into account, rather than focusing just on the present, drives us toward novelty."

Chapter 3: Sorting

In a full laundry hamper, to match 20 pairs of socks, you would have to dive your hand into the pile 19 times on average to find 1 first match, then 17 more times to match the second pair, resulting in 110 times to match all 20. For these kinds of problems, **Sorting Theory** offers solutions. Since sorting algorithms are highly sensitive to the number of items (e.g. the number of socks to match), first try reducing the scale (e.g. by doing laundry more often). Computer science measures such worse-case scenarios with **Big-O Notation**, which counts the number of operations that an algorithm executes. O(n) would mean that for n socks, you'd have to reach into the hamper n times.

Imagine you are organizing a bookshelf. Naturally, you may wish to simply scan the shelf and alphabetize as you go. This method is called **Bubble Sort** and takes place in quadratic time, or O(n^2), where n represents the