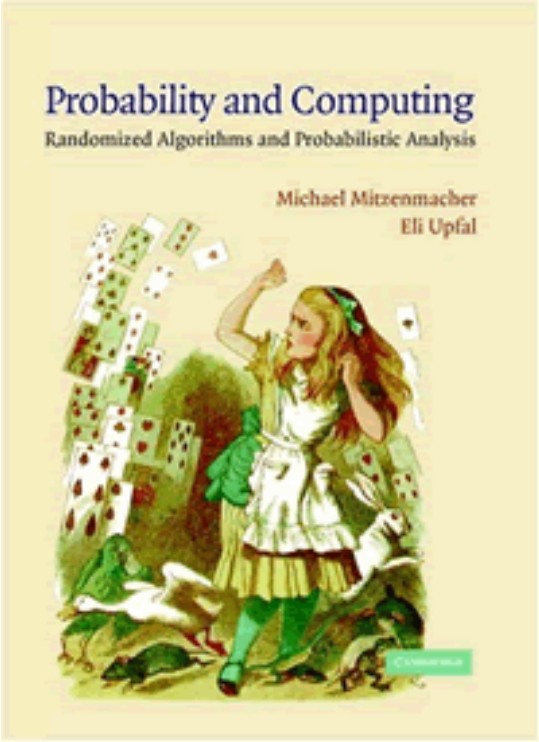
ΕΦΑΡΜΟΣΜΕΝΕΣ ΠΙΘΑΝΟΤΗΤΕΣ ΚΑΙ ΠΡΟΣΟΜΟΙΩΣΗ
Γ. Κοντογιάννης
3 Οκτωβρίου 2017
Φυλλάδιο #5

Probability and Computing: Randomized Algorithms and Probabilistic Analysis

Michael Mitzenmacher Eli Upfal
Harvard University Brown University



Probability and Computing

Randomization and probabilistic techniques play an important role in modern computer science, with applications ranging from combinatorial optimization and machine learning to communication networks and secure protocols.

This textbook is designed to accompany a one- or two-semester course for advanced undergraduates or beginning graduate students in computer science and applied mathematics.

It gives an excellent introduction to the probabilistic techniques and paradigms used in the development of probabilistic algorithms and analyses. It assumes only an elementary background in discrete mathematics and gives a rigorous yet accessible treatment of the material, with numerous examples and applications.

The first half of the book covers core material, including random sampling, expectations, Markov's inequality, Chebyshev's inequality, Chernoff bounds, balls-and-bins models, the probabilistic method, and Markov chains. In the second half, the authors delve into more advanced topics such as continuous probability, applications of limited independence, entropy, Markov chain Monte Carlo methods, coupling, martingales, and balanced allocations. With its comprehensive selection of topics, along with many examples and exercises, this book is an indispensable teaching tool.

Michael Mitzenmacher is John L. Loeb Associate Professor in Computer Science at Harvard University. He received his Ph.D. from the University of California, Berkeley, in 1996. Prior to joining Harvard in 1999, he was a research staff member at Digital Systems Research Laboratory in Palo Alto. He has received an NSF CAREER Award and an Alfred P. Sloan Research Fellowship. In 2002, he shared the IEEE Information Theory Society "Best Paper" Award for his work on error-correcting codes.

Eli Upfal is Professor and Chair of Computer Science at Brown University. He received his Ph.D. from the Hebrew University, Jerusalem, Israel. Prior to joining Brown in 1997, he was a research staff member at the IBM research division and a professor at the Weizmann Institute of Science in Israel. His main research interests are randomized computation and probabilistic analysis of algorithms, with applications to optimization algorithms, communication networks, parallel and distributed computing, and computational biology.

Preface

Why Randomness?

Why should computer scientists study and use randomness? Computers appear to behave far too unpredictably as it is! Adding randomness would seemingly be a disadvantage, adding further complications to the already challenging task of efficiently utilizing computers. Science has learned in the last century to accept randomness as an essential component in modelling and analyzing nature. In physics, for example, Newton's laws led people to believe that the universe was a deterministic place; given a big enough calculator and the appropriate initial conditions, one could determine the location of planets years from now. The development of quantum theory suggests a rather different view; the universe still behaves according to laws, but the backbone of these laws is probabilistic.

"God does not play dice with the universe" was Einstein's anecdotal objection to modern quantum mechanics. Nevertheless, the prevailing theory today for subparticle physics is based on random behavior and statistical laws, and randomness plays a significant role in almost every other field of science ranging from genetics and evolution in biology to modelling price fluctuations in a free-market economy.

Computer science is no exception. From the highly theoretical notion of probabilistic theorem proving to the very practical design of PC Ethernet cards, randomness and probabilistic methods play a key role in modern computer science. The last two decades have witnessed a tremendous growth in the use of probability theory in computing. Increasingly more advanced and sophisticated probabilistic techniques have been developed for use within broader and more challenging computer science applications.

In this book, we study the fundamental ways in which randomness comes to bear on computer science: randomized algorithms and the probabilistic analysis of algorithms.

***Randomized algorithms**:* Randomized algorithms are algorithms that make random choices during their execution. In practice, a randomized program would use values generated by a random number generator to decide the next step at several branches of its execution. For example, the protocol implemented in an Ethernet card uses random numbers to decide when it next tries to access the shared Ethernet communication medium. The randomness is useful for breaking symmetry, preventing different cards from repeatedly accessing the medium at the same time. Other commonly used applications of randomized algorithms include Monte Carlo simulations and primality testing in cryptography. In these and many other important applications, randomized algorithms are significantly more efficient than the best known deterministic solutions. Furthermore, in most cases the randomized algorithms are also simpler and easier to program.

These gains come at a price; the answer may have some probability of being incorrect, or the efficiency is guaranteed only with some probability. Although it may seem unusual to design an algorithm that may be incorrect, if the probability of error is sufficiently small then the improvement in speed or memory requirements may well be worthwhile.

***Probabilistic analysis of algorithms**:* Complexity theory tries to classify computation problems according to their computational complexity, in particular distinguishing between easy and hard problems. For example, complexity theory shows that the Traveling Salesmen problem is NP-hard. It is therefore very unlikely that there is an algorithm that can solve any instance of the Traveling Salesmen problem in time that is subexponential in the number of cities. An embarrassing phenomenon for the classical worst-case complexity theory is that

the problems it classifies as hard to compute are often easy to solve in practice. Probabilistic analysis gives a theoretical explanation for this phenomenon. Although these problems may be hard to solve on some set of pathological inputs, on most inputs (in particular, those that occur in real-life applications) the problem is actually easy to solve. More precisely, if we think of the input as being randomly selected according to some probability distribution on the collection of all possible inputs, we are very likely to obtain a problem instance that is easy to solve, and instances that are hard to solve appear with relatively small probability. Probabilistic analysis of algorithms is the method of studying how algorithms perform when the input is taken from a well-defined probabilistic space. As we will see, even NP-hard problems might have algorithms that are extremely efficient on almost all inputs.

Probability and Algorithms

The National Academies Press, 1992

Βιβλίο online στη σελίδα

http://fermat.nap.edu/books/0309047765/html/

Probability and Algorithms (1992)
Commission on Physical Sciences, Mathematics, and Applications

Panel on Probability and Algorithms
Committee on Applied and Theoretical Statistics
Board on Mathematical Sciences
Commission on Physical Sciences, Mathematics, and Applications
National Research Council
National Academy Press
Washington, D.C.
1992

PANEL ON PROBABILITY AND ALGORITHMS
J. MICHAEL STEELE,
University of Pennsylvania, Chair
DAVID ALDOUS,
University of California at Berkeley
DIMITRIS J. BERTSIMAS,
Massachusetts Institute of Technology
EDWARD G. COFFMAN,
AT&T Bell Laboratories
DORIT HOCHBAUM,
University of California at Berkeley
MICHA HOFRI,
University of Houston
JEFFREY C. LAGARIAS,
AT&T Bell Laboratories
SCOTT T. WEIDMAN, Senior Staff Officer

Preface

The Panel on Probability and Algorithms was constituted by the National Research Council in 1991 and charged with writing a report surveying both the topic of probabilistic algorithms, where randomization is a part of the internal calculation, and the probabilistic analysis of algorithms, in which one uses a probability model to deepen the understanding of how an algorithm functions in practice. Probabilistic algorithms—simulated annealing is one example—incorporate randomness into the underlying logic. For problems with huge solution spaces, or those where deterministic models of the processes involved do not exist, such algorithms have been very exciting developments. Probabilistic algorithms can solve certain problems faster than is possible by any deterministic algorithm. The probabilistic analysis of algorithms is a refinement of worst-case analysis, which is often too pessimistic compared to the performance of algorithms in actual practice.

Both uses of probability show tremendous promise, yet the research opportunities outnumber the people available to explore them. The panel hopes that more researchers, especially young researchers, will be intrigued by the possibilities and attracted to this field.

The panel gratefully acknowledges the substantial contributions of its contributing authors, Joan Feigenbaum, David Johnson, George Lueker, Bruce Maggs, Vijaya Rarnachandran, Ron Shamir, Peter Shor, and John Tsitsiklis. These colleagues greatly strengthened the report with their expertise. The contributions of Hans Oser, who served for a time as the project staff officer, are also appreciated.

Chapter 1. Introduction [μέρος, με μικρές αλλαγές]

By *J. Michael Steele*, University of Pennsylvania and *David Aldous*, University of California at Berkeley

The theory of algorithms has undergone an extraordinarily vigorous development over the last 20 years, and probability theory has emerged as one of its most vital partners. University courses, research monographs, and specialized journals have been developed to serve this partnership, yet there is still a need to communicate the progress in this area to the wider audience of computer scientists, mathematicians, and individuals who have a stake in the contributions that mathematical research can make to technology.

The main purpose of this report is to give an understanding of the power that comes from applying probability in the theory of algorithms, but an equally essential aim is to point out the *variety* of ways in which probability plays a role. One useful step in understanding this variety comes from making a clear distinction between the subject of *probabilistic algorithms* and the subject of *probabilistic analysis of algorithms*. Confusion sometimes arises over what methods are properly called "probabilistic (or randomized) algorithms"—and indeed there are some grey areas. Still, if one does not press for too fine a point, there are considerable organizational and conceptual benefits in drawing the distinction between probabilistic algorithms and the probabilistic analysis of a (possibly deterministic) algorithm.

One common distinction is that probabilistic algorithms, unlike deterministic ones, make random choices when computing. They are commonly referred to as "coin-flipping algorithms." Such

algorithms will produce (possibly) different results for the same problem when posed in different circumstances. On the other hand, the probabilistic analysis of an algorithm incorporates randomness into the *data* processed by an algorithm. Properly speaking, we are considering the pair (*algorithm, problem instance*) and probabilistically exploring the algorithm behavior over a large variety of problem instances. Typically, the analyst can make statements about the probability of selecting a particular instance, or focus attention on the distribution of suitable variables that describe the problem instance. The task is then to relate the algorithm performance to these variables.

This introductory chapter provides several simple illustrations of the distinction between the design of probabilistic algorithms and the probabilistic analysis of algorithms.

1.1.1 Everyday Examples of Probabilistic Algorithms

Before developing more serious examples that require detailed mathematical description, it seems useful to provide a couple of everyday analogies. Mathematics often speaks best for itself, and one should not read too much into analogies, but because there are important uses of probabilistic algorithms that can be explained without any technical prerequisites, it seems appropriate to look at them early on.

We have all had the experience of walking along a narrow path and encountering someone who is approaching on the same side of the path. As the individuals draw closer, one moves to the other side of the path in order to let the other person pass, but often the other person does the same simultaneously. This little shuffle continues one or two more times before the two people end up on opposite sides of the path and can pass each other.

Although people resolve such difficulties with little more cost than a moment's embarrassment, the analogous situation is more serious when packets of information end up vying for the use of the same place at the same time in a communication channel. When one tries to program machines to avoid such deadlocks, there are decided drawbacks to most deterministic rules. The dogged consistency of machines is such that the shuffling from side to side can go on unabated until an outside action halts the dance. This is an unacceptable state of affairs that one ought to be able to avoid through thoughtful design.

One theme of this report is that a natural course of action when one is faced by the disadvantages of purely deterministic rules is to introduce some sort of randomness into the protocol. Each process could continue to follow its own **randomized** rule without regard for the rule being followed by the other process, and the eternal dance would be avoided. This simple idea is at the heart of many of the communication protocols in use in the world today.

The larger idea onto which this flavor of application can be attached might be called "randomization to avoid special coincidences." Mathematicians can find analogs of this phenomenon in many other fields, and, as more technical examples will illustrate later, there are more profound consequences to the idea than are clone justice to by this first quick, everyday example.

Our second everyday example involves opinion polls. These are so widely discussed and so well understood that it is not easy to evoke a proper awareness that there is actually quite a bit of magic in the technology—enough so that there is still cutting-edge research in computer science related to the subject. Still, in order not to get ahead of our story, let us first consider a question

that is typical of the sort addressed by Gallup and other famous pollsters: What percentage of the U.S. voting population feels that the recent pay raise voted for itself by the U.S. Senate was well deserved? A fact that was completely unavailable to the founding fathers, but which is much a part of modern political life, is that one can obtain a useful answer to the question at a cost that is a small fraction of the cost of conducting a census of the whole voting populace.

One point that deserves to be underscored in this example is that the precise question that has been posed has nothing a priori to do with probability. If we are totally faithful to the precise phrasing of the question, we have to agree that it is one that can be answered only by a complete census as long as one insists on an *exact* answer. Still, a useful answer need not be perfectly exact, and the full complement of political insight is likely to be gained by knowing the answer to within plus or minus two percentage points. This fact is easily seen and accepted in this context, yet when one reviews the theory of algorithms one finds that tremendous effort is often expended to get exact answers where approximate ones might serve just as well. Thus, willingness to accept approximation seems to be one door for probability to enter. More surprising mathematical examples to be considered below will show that this is not the only door.

Still, in the context of our polling question, an approximation dearly serves the policy purpose just as well as a complete census. If one selects a random sample of about 2,500 from the target population and asks, "Do you agree that the pay raise that the U.S. Senate gave itself is well deserved?", the percentage of individuals who say yes provides an estimate of the percentage of the whole populace that would answer in the same way. It is a consequence of elementary probability theory that our estimate will be correct to within ±2 percent on better than 95 percent of the occasions on which such a sampling experiment is conducted.

This is a familiar tale, even if told a bit more precisely than usual, and the reader may legitimately ask why this old tale is being told here. The retelling is not occasioned to celebrate polling, which certainly draws enough of its own attention. The point is not even to recall the often missed subtlety that the required sample size depends only on the precision required, and not on the size of the population about which the inference is being made; though it is interesting that the senators from Rhode Island find it as costly to obtain information about the voting populace of that state as it is for the National Committee to get the corresponding information about the nation as whole.

The point is rather to see that the sampling techniques of political pollsters are actually probabilistic algorithms; one uses exogenous randomization to obtain an approximation to a problem that would be prohibitively expensive to answer exactly. The key point for us is not the approximation issue. Approximation is a common but not inevitable feature of probabilistic algorithms. The key point is that one pours in randomness that was not there to begin with. In the pollster's case the randomness that was added was that used in making the random selection. What was purchased at the price of this extra complication was the applicability of the probability theory that enabled us to quantify the quality of our solution. These are features that one finds throughout the theory of probabilistic algorithms.

To be sure, there are differences in flavor between the pollsters' techniques and those applied in computer science. The pollsters' methods are applied in a social rather than a computational context, and, perhaps as a result, the process looks unsophisticated, or even straightforward. Individuals will sometimes disagree, but it seems useful to hold that the pollsters' techniques are just as much a probabilistic algorithm as any being studied in computer science. The difference of context does not alter the logical structure, and the perceived difference in sophistication comes

in good measure from the fact that our example did not go into any subtleties such as how one might really draw the sample and whether some trickier sampling scheme might do better than a uniform random sample.

This pollster problem completes the second of our two everyday examples. Both were illustrations of randomization applied in algorithms. In one case the randomization offered escape from the "special situation" traps into which deterministic processes can fall. In the second case, we gained great savings of cost at the price of accepting an approximation, and we were able to quantify the quality of our approximation because we were able to add just the kind of extra randomness to which a compelling theory could be applied.

[...]

1.1.5 Random Constructions

A surprising connection between mathematical probability and the theory of algorithms is the idea of proving mathematical results about random objects by studying the behavior of an algorithm for constructing the random object. We illustrate this with the classical topic of random permutations. Suppose we want to generate, inductively on n, the uniform random permutation of n objects into n positions. Picturing the objects as physical files in a file cabinet, the natural update algorithm for adding the nth object is: pick p uniformly on 1, . . . , p, move objects at positions $i = p, p+1, \ldots, n$ to positions $i + 1$, and put object n at position p. In a computer the natural update algorithm is: put object n at position n, pick p uniformly on 1, . . . , p, and switch the objects at positions n and p.

Though the latter algorithm is essentially the standard optimal algorithm for computer simulation of a random permutation, here is a more complicated algorithm that turns out to be useful for mathematical purposes:

The Chinese restaurant process. Mathematicians at a conference agree to dine at a Chinese restaurant. They arrive separately, and so the ith arrival has i choices of where to sit: next (clockwise, say) to one of the i-1 mathematicians already present, or at an unoccupied table. Suppose the choice is random and uniform. After n arrivals, the pattern of mathematicians (labelled by arrival order) at tables can be regarded as the cycle representation of a permutation, and it is easy to see that the sequential uniform random choices create a uniform random permutation.

Suppose we want to study the random number C_n of cycles in a random permutation. In terms of the Chinese restaurant algorithm, C_n is just the number of occupied tables. But the ith arrival has chance $1/i$ of starting a new table, so without calculation we can see that the expected value of C_n is simply $\sum_{i=1}^{n} 1/i$.

1.2 Probabilistic Analysis of Algorithms: The Assignment Problem

Probabilistic analysis is inevitably mathematical, so no everyday example can help illustrate this second of the great gates through which probability theory enters the theory of algorithms. Still, for our first such example, we can use a task that often arises as a module in larger computational problems and that is evocatively expressed in language that reflects its origins in industrial engineering. It requires a slight increase in technical level over our everyday examples, but it is

still friendly and non-technical. An additional benefit of the problem is that its analysis offers several surprises that have only recently been discovered.

We suppose that we have a set of n jobs labelled $\{1, 2, \ldots, n\}$ and a set of n machines on which to perform the jobs that are similarly labelled. We further suppose that there is a matrix of numbers c_{ij} that describe the cost of doing job i on machine j. A natural and important computational task is to specify a one-to-one assignment of jobs to machines that minimizes the total cost. In other words, the task is to determine a permutation σ that minimizes

$$A(\sigma) = \sum_{i=1}^{n} c_{i\sigma(i)} \ .$$

This is called the **assignment problem**, and one can see how it could come about easily on its own or in the context of a larger computational task.

One natural way to try to solve this problem is to use a "greedy" strategy. There are two natural ways to proceed, so we will consider the simplest one first. We look at job 1 and assign it to machine j, where j is chosen to minimize c_{1j} over all $1 \le j \le n$. We then successively assign jobs $2, 3, \ldots, n$ by choosing at each step to take the least costly among the unassigned machines.

There is a slightly more sophisticated algorithm that takes a more globally greedy perspective. For the first assignment, one chooses the assignment $i \to j$, where (i, j) are chosen to minimize c_{ij} over all n^2 possible choices. The second assignment then chooses the cheapest possibility from the remaining set of $(n-1)^2$ pairs of feasible assignments. One then continues in this globally greedy way until one has a complete assignment of jobs to machines. The natural problem in this context is to determine which of the two methods is better. Another compelling problem is to determine if there are still other methods that do substantially better than these two greedy procedures.

One way to approach these problems is to assume some probability model on the input data c_{ij} and to calculate appropriate measures of performance under this model. An issue that emerges at this point is that for algorithms that do not return an exact solution, there are two compelling dimensions along which to measure performance: the quality of the solution obtained and the amount of time needed to obtain that solution. For the simple and global greedy algorithms for the assignment problem, the running time analysis does not depend on probability theory, so we will consider it first.

Because one can find the smallest element in a list of k items in time that is bounded by a constant times k, the running time of the first algorithm is bounded by $c[n + (n-1) + (n-2) + \ldots + 1] = O(n^2)$. The second algorithm requires a little more knowledge to implement efficiently, but for people who have had some exposure to the subject of data structures there are some off-the-shelf tools that come to mind almost immediately. One such tool is a data structure called a **heap**. Given m items that have a total order, one can build this structure in time $O(m)$, and it has the remarkable property that one can delete the largest item in the heap at a cost bounded by $c \log m$ and be left with a new heap that is now of size $m-1$. There are other structures besides heaps that have this property, but unless one is ready to get down to coding, there is no need to do more than note that there is some *abstract* data structure that can be built at the specified cost and that permits deletion at the specified cost.

We can now go about measuring the running time cost of our greedy algorithms. If we put the n^2 costs c_{ij} into a heap at a cost of $O(n^2)$, then even if we have to delete all the items in the heap to build our assignment, the total time used is still only $O(n^2 \log n)$. The bottom line of these two running time analyses is that both of the algorithms are quite practical, just as we expected them to be, and the global greedy algorithm is more costly in time to the tune of a factor of log n. This second result is also much as one might have expected.

A bigger surprise comes when one looks at the quality of the solutions that are obtained by the two algorithms. Here, a natural way to proceed is to make some probabilistic assumptions about the cost c_{ij}. One is not likely to find any assumption that is more natural than to take the c_{ij} to be independent, identically distributed random variables. Further, because we are just looking for *some* computable feature of merit that casts light on the quality of the assignments we obtain, we might as well go the rest of the way in making our model easy by assuming that the c_{ij} are uniformly distributed on [0, 1].

This setup does indeed make it easy to compute the expected value of the total assignment cost A_n that is obtained using the simple greedy algorithm. Because the expected value of the minimum of k independent, uniformly distributed random variables is exactly equal to $1/(k + 1)$, we see

$$E(A_n) = 1/(n+1) + 1/n + 1/(n-1) + \ldots + 1/2 \sim \log_e(n).$$

The analysis of the expected value of A'_n, the cost of the assignment obtained under the global greedy method, is a little bit more difficult to obtain. The problem is that one loses all independence in the model after the very first step. One can nevertheless show by setting up a simple integral equation based on dynamic programming that one again has that

$$E(A'_n) \sim \log_e(n).$$

The interesting point here is that even though the global greedy algorithm takes a little more time to compute and seems to be a more powerful strategy, the value of the assignment that it yields is typically no better than that one gets by the simple greedy strategy.

Before leaving this example, we note that these heuristics fall short of providing the deepest understanding of $E(A_n^{OPT})$, where $E(A_n^{OPT})$ denotes the least cost of any assignment. Karp in 1987 showed by a remarkable argument that

$$E(A_n^{OPT}) \leq 2,$$

and the insight provided by Karp's proof has been shown by Dyer to apply to many other problems that can be expressed as linear programs. Finally, Mézard and Parisi offered intriguing calculations based on methods of statistical mechanics that suggest

$$E(A_n^{OPT}) \sim \pi^2/6, \text{ as } n \to \infty,$$

and this was only recently proved to be true, using hard and very novel probabilistic techniques.