Randomized Algorithms

RAJEEV MOTWANI

Department of Computer Science, Stanford University, Stanford, California

PRABHAKAR RAGHAVAN

IBM Almaden Research Center, San Jose, California

Randomized algorithms, once viewed as a tool in computational number theory, have by now found widespread application. Growth has been fueled by the two major benefits of randomization: simplicity and speed. For many applications a randomized algorithm is the fastest algorithm available, or the simplest, or both.

A randomized algorithm is an algorithm that uses random numbers to influence the choices it makes in the course of its computation. Thus its behavior (typically quantified as running time or quality of output) varies from one execution to another even with a fixed input. In the analysis of a randomized algorithm we establish bounds on the expected value of a performance measure (e.g., the running time of the algorithm) that are valid for *every* input; the distribution of the performance measure is on the random choices made by the algorithm based on the random bits provided to it.

Caveat: The analysis of randomized algorithms should be distinguished from the *probabilistic* or *average-case* analysis of an algorithm, in which it is assumed that the *input* is chosen from a probability distribution. In the latter case, the analysis would typically imply only that the algorithm is good for *most* inputs but not for all.

HISTORY AND SOURCES

The roots of randomized algorithms can be traced back to *Monte Carlo methods* used in numerical analysis, statistical physics, and simulation. In complexity theory, the notion of a probabilistic Turing machine was proposed by de Leeuw et al. [1955] and was further explored in the pioneering work of Rabin [1963] and Gill [1977]. The earliest examples of concrete randomized algorithms appeared in the work of Berlekamp [1970], Rabin [1976], and Solovay and Strassen [1977]. Rabin [1976] explicitly proposed randomization as an algorithmic tool using as examples problems in computational geometry and in number theory. At about the same time, Solovay and Strassen [1977] presented a randomized primality-testing algorithm; these were predated by the randomized polynomial-factoring algorithm presented by Berlekamp [1970].

Since then, an impressive array of techniques for devising and analyzing randomized algorithms have been developed. The reader may refer to Karp [1991], Maffioli et al. [1985], and Welsh [1983] for recent surveys of the research into randomized algorithms. The probabilistic (or "average-case") analysis of algorithms (sometimes also called "distributional complexity") is surveyed by Johnson [1984a], and this is contrasted with randomized algorithms in his following bulletin [1984b].

The work of R. Motwani was supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

Copyright © 1996, CRC Press.

The recent book by the authors [Motwani and Raghavan 1995] gives a comprehensive introduction to randomized algorithms.

PARADIGMS FOR RANDOMIZED ALGORITHMS

In spite of the multitude of areas in which randomized algorithms find application, a handful of general principles underlie almost all of them. Using the summary in Karp [1991], we present these principles in the following.

Foiling an Adversary. In the classical worst-case analysis of deterministic algorithms, a lower bound is established on the running time of algorithms by postulating an "adversary" that constructs an input on which the algorithm fares poorly. The input thus constructed may be different for each deterministic algorithm. With a game-theoretic interpretation of the relationship between an algorithm and an adversary, we can view a randomized algorithm as a probability distribution on a set of deterministic algorithms. (This observation underlies Yao's [1977] adaptation of von Neumann's Mini-Max Theorem in game theory into a technique for establishing limits on the performance improvements possible via the use of a randomized algorithm.) Although the adversary may be able to construct an input that foils one (or a small fraction) of the deterministic algorithms in the set, it may be impossible to devise a single input that is likely to defeat a randomly chosen algorithm. For example, consider a uniform binary AND-OR tree with n leaves. Any deterministic algorithm that evaluates such a tree can be forced to read the Boolean values at every one of the n leaves. However, there is a simple randomized algorithm [Snir 1985] for which the expected number of leaves read on any input is $O(n^{0.794})$.

Random Sampling. A pervasive theme in randomized algorithms is the idea that a *small* random sample from a population is representative of the population as a whole. Because computations involving small samples are inexpensive, their properties can be used to guide the computations of an algorithm attempting to determine some feature of the entire population. For instance, a simple randomized algorithm [Floyd and Rivest 1975] based on sampling finds the kth largest of n elements in $1.5n + o(n)$ comparison steps, with high probability. In contrast, it is known that any deterministic algorithm must make at least $2n$ comparisons in the worst case.

Abundance of Witnesses. Often a computational problem can be reduced to finding a witness or a certificate that would efficiently verify an hypothesis. (For example, to show that a number is not prime, it suffices to exhibit any nontrivial factor of that number.) For many problems, the witness lies in a search space that is too large to be searched exhaustively. However, if the search space were to contain a relatively large number of witnesses, a randomly chosen element is likely to be a witness. Further, independent repetitions of the sampling reduce the probability that a witness is not found on any of the repetitions. The most striking examples of this phenomenon occur in number theory. Indeed, the problem of testing a given integer for primality has no known deterministic polynomial-time algorithm. There are, however, several randomized polynomial-time algorithms [Solovay and Strassen 1978; Rabin 1980] that will, on any input, correctly perform this test with high probability.

Fingerprinting and Hashing. A fingerprint is the image of an element from a (large) universe under a mapping into another (smaller) universe. Fingerprints obtained via random mappings have many useful properties. For example, in pattern-matching applications [Karp and Rabin 1987] it can be shown that two strings are likely to be identical if their fingerprints are identical;

comparing the short fingerprints is considerably faster than comparing the strings themselves. Another example is *hashing* [Carter and Wegman 1979], where the elements of a set S (drawn from a universe U) are stored in a table of size linear in $|S|$ (even though $|U| \gg |S|$) with the guarantee that the expected number of elements in S mapped to a given location in the table is $O(1)$. This leads to efficient schemes for deciding membership in S. Random fingerprints have found a variety of applications in generating pseudorandom numbers and complexity theory (for instance, the verification of algebraic identities [Freivalds 1977]).

Random Reordering. A large class of problems has the property that a relatively naive algorithm A can be shown to perform extremely well provided the input data is presented in a random order. Although A may have poor worst-case performance, randomly reordering the input data ensures that the input is unlikely to be in one of the orderings that is pathological for A. The earliest instance of this phenomenon can be found in the behavior of the Quicksort algorithm [Hoare 1962]. The random reordering approach has been particularly successful in tackling problems in data structures and computational geometry. For instance, there are simple algorithms for computing the convex hull of n points in the plane that process the input one point at a time. Such algorithms are doomed to take $\Omega(n^2)$ steps if the input points are presented in an order determined by an adversary; however, if they are processed in random order, the running time drops to $O(n \log n)$. The book by Mulmuley [1993] gives an excellent overview of randomized geometric algorithms.

Load Balancing. When we must choose between different resources (such as links in a communication network or when assigning tasks to parallel processors), randomization can be used to "spread" the load evenly among the resources. This paradigm has found many interesting applications in parallel and distributed computing where resource utilization decisions have to be made locally, without global knowledge. Consider packet routing in an n-node butterfly network. It is known that $\Omega(\sqrt{n})$ steps are required by any deterministic oblivious algorithm (a class of simple routing algorithms in which the route followed by a packet is independent of the routes of other packets). In contrast, based on ideas of Valiant [1982], it has been shown [Aleliunas 1982; Upfal 1984] that there is a randomized algorithm that terminates in $O(\log n)$ steps with high probability.

Rapidly Mixing Markov Chains. In counting problems, the goal is to determine the number of combinatorial objects with a specified property. When the space of objects is large, an appealing solution is the use of the Monte Carlo approach of determining the number of desired objects in a random sample of the entire space. In a number of cases, it can be shown that picking a uniform random sample is as difficult as the counting problem itself. A particularly successful technique for dealing with such problems is to generate near-uniform random samples by defining a Markov chain on the elements of the population, and showing that a short random walk using this Markov chain is likely to sample the population uniformly. This method is at the core of a number of algorithms used in statistical physics [Sinclair 1992]. Examples include algorithms for estimating the number of perfect matchings in a graph [Jerrum and Sinclair 1989].

Isolation and Symmetry Breaking. In computing on asynchronous distributed processors, it is often necessary for a collection of processors to break a deadlock or a symmetry and make a common choice. Randomization is a powerful tool in such deadlock-avoidance. For example, see the protocol for choice coordination due to Rabin [1982]. Similarly, in

parallel computation, often a problem has many feasible solutions and so it becomes important to ensure that the different processors are working towards finding the same solution. This involves isolating a specific solution out of the space of all feasible solutions without actually knowing any single element of the solution space. One clever randomized strategy for *isolation* chooses a random ordering on the feasible solutions and then requires the processors to focus on finding the solution of the lowest rank. This idea has proved to be critical in devising efficient parallel algorithms for finding a perfect matching in a graph [Mulmuley et al. 1987].

Probabilistic Methods and Existence Proofs. The probabilistic method attempts to establish the existence of a specific type of combinatorial object by arguing that a random object from a suitably defined universe has the desired property with nonzero probability. Usually this method gives no clue on actually finding such an object. This method is sometimes used to guarantee the existence of an algorithm for solving a problem; we thus know that the algorithm exists, but have no idea what it looks like or how to construct it. The book by Alon and Spencer [1992] gives an excellent overview of this subject.

REFERENCES

ALELIUNAS, R. 1982. Randomized parallel communication. In *ACM-SIGOPS Symposium on Principles of Distributed Systems,* 60–72.

ALON, N. AND SPENCER, J. 1992. *The Probabilistic Method.* Wiley, New York.

BERLEKAMP, E. R. 1970. Factoring polynomials over large finite fields. *Math. Comput. 24,* 713–735.

CARTER, J. L. AND WEGMAN, M. N. 1979. Universal classes of hash functions. *J. Comput. Syst. Sci. 18,* 2, 143–154.

DE LEEUW, K., MOORE, E. F., SHANNON, C. E., AND SHAPIRO, N. 1955. Computability by probabilistic machines. In *Automata Studies,* C. E. Shannon and J. McCarthy, Eds., Princeton University Press, Princeton, NJ, 183–212.

FLOYD, R. W. AND RIVEST, R. L. 1975. Expected time bounds for selection. *Commun. ACM 18,* 165–172.

FREIVALDS, R. 1977. Probabilistic machines can use less running time. In *Information Processing 77, Proceedings of IFIP Congress 77,* B. Gilchrist, Ed., (Aug.), North-Holland, Amsterdam, 839–842.

GILL, J. 1977. Computational complexity of probabilistic Turing machines. *SIAM J. Comput. 6,* 4 (Dec.), 675–695.

HOARE, C. A. R. 1962. Quicksort. *Comput. J. 5,* 10–15.

JERRUM, M. R. AND SINCLAIR, A. 1989. Approximating the permanent. *SIAM J. Comput. 18,* 6 (Dec.), 1149–1178.

JOHNSON, D. S. 1984a. The NP-completeness column: An ongoing guide. *J. Algorithms 5,* 284–299.

JOHNSON, D. S. 1984b. The NP-completeness column: An ongoing guide. *J. Algorithms 5,* 433–447.

KARP, R. M. 1991. An introduction to randomized algorithms. *Discrete Appl. Math. 34,* 165–201.

KARP, R. M. AND RABIN, M. O. 1987. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev. 31* (March), 249–260.

MAFFIOLI, F., SPERANZA, M. G., AND VERCELLIS, C. 1985. Randomized algorithms. In *Combinatorial Optimization: Annotated Bibliographies,* M. O'hEigertaigh, J.K. Lenstra, and A.H.G. Rinooy Kan, Eds., Wiley, New York, 89–105.

MOTWANI, R. AND RAGHAVAN, P. 1995. *Randomized Algorithms.* Cambridge University Press, New York. World-Wide Web information at http://www.cup.org/Reviews&blurbs/RanAlg/RanAlg.html.

MULMULEY, K. 1993. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York.

MULMULEY, K., VAZIRANI, U. V., AND VAZIRANI, V. V. 1987. Matching is as easy as matrix inversion. *Combinatorica 7,* 105–113.

RABIN, M. O. 1982. The choice coordination problem. *Acta Inf. 17,* 121–134.

RABIN, M. O. 1980. Probabilistic algorithm for testing primality. *J. Number Theory 12,* 128–138.

RABIN, M. O. 1976. Probabilistic algorithms. In *Algorithms and Complexity, Recent Results and New Directions,* J.F. Traub, Ed., Academic Press, New York, 21–39.

RABIN, M. O. 1963. Probabilistic automata. *Inf. Control 6,* 230–245.

SINCLAIR, A. 1992. *Algorithms for Random Generation and Counting: A Markov Chain Approach.* Progress in Theoretical Computer Science. Birkhauser, Boston.

SNIR, M. 1985. Lower bounds on probabilistic linear decision trees. *Theor. Comput. Sci. 38,* 69–82.

SOLOVAY, R. AND STRASSEN, V. 1977. A fast Monte-Carlo test for primality. *SIAM J. Comput. 6,* 1 (March), 84–85. See also *SIAM J. Comput. 7,* 1 (Feb.), 1978, 118.

UPFAL, E. 1984. Efficient schemes for parallel communication. *J. ACM 31,* 507–517.

VALIANT, L. G. 1982. A scheme for fast parallel communication. *SIAM J. Comput. 11,* 350–361.

WELSH, D. J. A. 1983. Randomised algorithms. *Discrete Appl. Math. 5,* 133–145.

YAO, A. C-C. 1977. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science,* 222–227.