1

Introduction

1.1 Overview of this Survey

Over the past few decades, *randomization* has become one of the most pervasive paradigms in computer science. Its widespread uses include:

Algorithm Design: For a number of important algorithmic problems, the most efficient algorithms known are randomized. For example:

- Primality Testing: This was shown to have a randomized polynomial-time algorithm in 1977. It wasn't until 2002 that a deterministic polynomial-time algorithm was discovered. (We will see this algorithm, but not its proof.)
- *Approximate Counting*: Many approximate counting problems (e.g., counting perfect matchings in a bipartite graph) have randomized polynomial-time algorithms, but the fastest known deterministic algorithms take exponential time.
- Undirected S-T Connectivity: This was shown to have a randomized logspace algorithm in 1979. It wasn't until 2005 that a deterministic logspace algorithm was discovered — using tools from the theory of pseudorandomness, as we will see.

- Perfect Matching: This was shown to have a randomized *polylogarithmic*-time parallel algorithm in the late 1970s. Deterministically, we only know polynomial-time algorithms.

Even in the cases where deterministic algorithms of comparable complexity were eventually found, the randomized algorithms remain considerably simpler and more efficient than the deterministic ones.

Cryptography: Randomization is central to cryptography. Indeed, cryptography is concerned with protecting secrets, and how can something be secret if it is deterministically fixed? For example, we assume that cryptographic keys are chosen at random (e.g., uniformly from the set of n-bit strings). In addition to the keys, it is known that often the cryptographic algorithms themselves (e.g., for encryption) must be randomized to achieve satisfactory notions of security (e.g., that no partial information about the message is leaked).

Combinatorial Constructions: Randomness is often used to prove the *existence* of combinatorial objects with a desired property. Specifically, if one can show that a randomly chosen object has the property with nonzero probability, then it follows that such an object must, in fact, exist. A famous example due to Erdős is the existence of *Ramsey graphs*: A randomly chosen n-vertex graph has no clique or independent set of size $2\log n$ with high probability. We will see several other applications of this "Probabilistic Method" in this survey, such as with two important objects mentioned below: expander graphs and error-correcting codes.

Though these *applications* of randomness are interesting and rich topics of study in their own right, they are not the focus of this survey. Rather, we ask the following:

Main Question: Can we reduce or even eliminate the use of randomness in these settings?

We have several motivations for doing this.

- *Complexity Theory*: We are interested in understanding and comparing the power of various kinds of computational resources. Since randomness is such a widely used resource,

we want to know how it relates to other resources such as time, space, and parallelism. In particular, we ask: *Can every randomized algorithm be derandomized with only a small loss in efficiency?*

- *Using Physical Random Sources*: It is unclear whether the real world has physical sources of perfect randomness. We may use sources that seem to have some unpredictability, like the low order bits of a system clock or thermal noise, but these sources will generally have biases and, more problematically, correlations. Thus we ask: What can we do with a source of biased and correlated bits?
- *Explicit Constructions*: Probabilistic constructions of combinatorial objects often do not provide us with efficient algorithms for using those objects. Indeed, the randomly chosen object often has a description that is exponential in the relevant parameters. Thus, we look for *explicit* constructions — ones that are deterministic and efficient. In addition to their applications, improvements in explicit constructions serve as a measure of our progress in understanding the objects at hand. Indeed, Erdős posed the explicit construction of near-optimal Ramsey graphs as an open problem, and substantial progress on this problem was recently made using the theory of pseudorandomness (namely randomness extractors).
- *Unexpected Applications*: In addition, the theory of pseudorandomness has turned out to have many applications to problems that seem to have no connection to derandomization. These include data structures, distributed computation, circuit lower bounds and completeness results in complexity theory, reducing interaction in interactive protocols, saving memory in streaming algorithms, and more. We will see some of these applications in this survey (especially the exercises).

The paradigm we will use to study the Main Question is that of *pseudorandomness*: efficiently generating objects that "look random" using little or no randomness.

Specifically, we will study four "pseudorandom" objects:

Pseudorandom Generators: A pseudorandom generator is an algorithm that takes as input a short, perfectly random *seed* and then returns a (much longer) sequence of bits that "looks random." That the bits output cannot be perfectly random is clear — the output is determined by the seed and there are far fewer seeds than possible bit sequences. Nevertheless, it is possible for the output to "look random" in a very meaningful and general-purpose sense. Specifically, we will require that no *efficient* algorithm can distinguish the output from a truly random sequence. The study of pseudorandom generators meeting this strong requirement originated in cryptography, where they have numerous applications. In this survey, we will emphasize their role in derandomizing algorithms.

Note that asserting that a function is a pseudorandom generator is a statement about something that efficient algorithms can't do (in this case, distinguish two sequences). But proving that efficient algorithms cannot compute things is typically out of reach for current techniques in theoretical computer science; indeed this is why the **P** vs. **NP** question is so hard. Thus, we will settle for conditional statements. An ideal theorem would be something like: "If $\mathbf{P} \neq \mathbf{NP}$, then pseudorandom generators exist." (The assumptions we make won't exactly be $\mathbf{P} \neq \mathbf{NP}$, but hypotheses of a similar flavor.)

Randomness Extractors: A randomness extractor takes as input a source of biased and correlated bits, and then produces a sequence of almost-uniform bits as output. Their original motivation was the simulation of randomized algorithms with sources of biased and correlated bits, but they have found numerous other applications in theoretical computer science. Ideally, extractors would be deterministic, but as we will see this proves to be impossible for general sources of biased and correlated bits. Nevertheless, we will get close — constructing extractors that are only "mildly" probabilistic, in that they use small (logarithmic) number of truly random bits as a seed for the extraction.

Expander Graphs: Expanders are graphs with two seemingly contradictory properties: they are sparse (e.g., having degree that is

a constant, independent of the number of vertices), but also "well-connected" in some precise sense. For example, the graph cannot be bisected without cutting a large (say, constant) fraction of the edges.

Expander graphs have numerous applications in theoretical computer science. They were originally studied for their use in designing fault-tolerant networks (e.g., for telephone lines), ones that maintain good connectivity even when links or nodes fail. But they also have less obvious applications, such as an $O(\log n)$-time algorithm for sorting in parallel.

It is not obvious that expander graphs exist, but in fact it can be shown, via the Probabilistic Method, that a random graph of degree 3 is a "good" expander with high probability. However, many applications of expander graphs need *explicit constructions*, and these have taken longer to find. We will see some explicit constructions in this survey, but even the state-of-the-art does not always match the bounds given by the probabilistic method (in terms of the degree/expansion tradeoff).

Error-Correcting Codes: Error-correcting codes are tools for communicating over noisy channels. Specifically, they specify a way to encode messages into longer, redundant codewords so that even if the codeword gets somewhat corrupted along the way, it is still possible for the receiver to decode the original message. In his landmark paper that introduced the field of coding theory, Shannon also proved the existence of good error-correcting codes via the Probabilistic Method. That is, a random mapping of n-bit messages to $O(n)$-bit codewords is a "good" error-correcting code with high probability. Unfortunately, these probabilistic codes are not feasible to actually use — a random mapping requires an exponentially long description, and we know of no way to decode such a mapping efficiently. Again, explicit constructions are needed.

In this survey, we will focus on the problem of *list decoding*. Specifically, we will consider scenarios where the number of corruptions is so large that unique decoding is impossible; at best one can produce a short list that is guaranteed to contain the correct message.

A Unified Theory: Each of the above objects has been the center of a large and beautiful body of research, but until recently these corpora

were largely distinct. An exciting development over the past decade has been the realization that all four of these objects are almost *the same* when interpreted appropriately. Their intimate connections will be a major focus of this survey, tying together the variety of constructions and applications that we will see.

The surprise and beauty of these connections has to do with the seemingly different nature of each of these objects. Pseudorandom generators, by asserting what efficient algorithms cannot do, are objects of complexity theory. Extractors, with their focus on extracting the entropy in a correlated and biased sequence, are information-theoretic objects. Expander graphs are of course combinatorial objects (as defined above), though they can also be interpreted algebraically, as we will see. Error-correcting codes involve a mix of combinatorics, information theory, and algebra. Because of the connections, we obtain new perspectives on each of the objects, and make substantial advances on our understanding of each by translating intuitions and techniques from the study of the others.

1.2 Background Required and Teaching Tips

The presentation assumes a good undergraduate background in the theory of computation, and general mathematical maturity. Specifically, it is assumed that the reader is familiar with basic algorithms and discrete mathematics as covered in [109], including some exposure to randomized algorithms; and with basic computational complexity including P, NP, and reductions, as covered in [366]. Experience with elementary abstract algebra, particularly finite fields, is helpful; recommended texts are [36, 263].

Most of the material in the survey is covered in a one-semester graduate course that the author teaches at Harvard University, which consists of 24 lectures of 1.5 hours each. Most of the students in that course take at least one graduate-level course in theoretical computer science before this one.

The exercises are an important part of the survey, as they include proofs of some key facts, introduce some concepts that will be used in later sections, and illustrate applications of the material to other topics.

Problems that are particularly challenging or require more creativity than most are marked with a star.

1.3 Notational Conventions

We denote the set of numbers $\{1,\ldots,n\}$ by $[n]$. We write $\mathbb{N}$ for the set of nonnegative integers (so we consider 0 to be a natural number). We write $S \subset T$ to mean that S is a subset of T (not necessarily strict), and $S \subsetneq T$ for S being a strict subset of T. An inequality we use often is $\binom{n}{k} \leq (ne/k)^k$. All logarithms are base 2 unless otherwise specified. We often use the convention that lowercase letters are the logarithm (base 2) of the corresponding capital letter (e.g., $N = 2^n$).

Throughout, we consider random variables that can take values in arbitrary discrete sets (as well as real-valued random variables). We generally use capital letters, e.g., X, to denote random variables and lowercase letters, e.g., x, to denote specific values. We write $x \xleftarrow{R} X$ to indicate that x is sampled according to X. For a set S, we write $x \xleftarrow{R} S$ to mean that x is selected uniformly at random from S. We use the convention that multiple occurrences of a random variable in an expression refer to the same instantiation, e.g., $\Pr[X = X] = 1$. The *support* of a random variable X is denoted by $\mathrm{Supp}(X) \stackrel{\mathrm{def}}{=} \{x : \Pr[X = x] > 0\}$. For an event E, we write $X|_E$ to denote the random variable X conditioned on the event E. For a set S, we write U_S to denote a random variable distributed uniformly over S. For $n \in \mathbb{N}$, U_n is a random variable distributed uniformly over $\{0,1\}^n$.

1.4 Chapter Notes and References

In this section, we only provide pointers to general surveys and textbooks on the topics covered, plus citations for specific results mentioned.

General introductions to the theory of pseudorandomness (other than this survey) include [162, 288, 393].

Recommended textbooks focused on randomized algorithms are [290, 291]. The specific randomized and deterministic algorithms

mentioned are due to [6, 13, 220, 236, 237, 267, 287, 314, 327, 369]; for more details see Section 2.6.

Recommended textbooks on cryptography are [157, 158, 238]. The idea that encryption should be randomized is due to Goldwasser and Micali [176].

The Probabilistic Method for combinatorial constructions is the subject of the book [25]. Erdős used this method to prove the existence of Ramsey graphs in [132]. Major recent progress on explicit constructions of Ramsey graphs was recently obtained by Barak, Rao, Shaltiel, and Widgerson [48] via the theory of randomness extractors.

The modern notion of a pseudorandom generator was formulated in the works of Blum and Micali [72] and Yao [421], motivated by cryptographic applications. We will spend most of our time on a variant of the Blum–Micali–Yao notion, proposed by Nisan and Wigderson [302], where the generator is allowed more running time than the algorithms it fools. A detailed treatment of the Blum–Micali–Yao notion can be found in [157].

Surveys on randomness extractors are [301, 352, 354]. The notion of extractor that we will focus on is the one due to Nisan and Zuckerman [303].

A detailed survey of expander graphs is [207]. The probabilistic construction of expander graphs is due to Pinsker [309]. The application of expanders to sorting in parallel is due to Ajtai, Komlós, and Szemerédi [10].

A classic text on coding theory is [282]. For a modern, computer science-oriented treatment, we recommend Sudan's lecture notes [380]. Shannon started this field with the paper [361]. The notion of list decoding was proposed by Elias [129] and Wozencraft [420], and was reinvigorated in the work of Sudan [378]. Recent progress on list decoding is covered in [184].