6.841: Advanced Complexity Theory Fall 2012

Lecture 10 (Avi Wigderson) — October 11, 2012

Prof. Dana Moshkovitz *Scribe: Daniel Fremont*

1 Overview

Randomness is an incredibly useful phenomenon, with applications in many fields. We can solve some problems using randomness exponentially faster than we know how to do deterministically. Do we need truly random bits for these algorithms? What if we don't have access to perfect randomness? Using results from the theory of computational pseudorandomness, we find that many algorithms can be made to work with very weak sources of randomness, and possibly with no randomness at all.

2 The Utility of Randomness

Randomness is useful in a very wide variety of settings. The most familiar example is its use in statistics to quickly gather aggregate information about a large population. For example, if you have 280 million people voting either red or blue, you can poll a random sample of 2,000 people. Then it can be shown (using the law of large numbers) that with probability > 0.99, the fraction of people voting red in the sample is within 2% of the fraction voting red in the whole population. Furthermore, the sample size of 2,000 gives this accuracy independent of the total population size. This is an extremely powerful method: deterministically, you would have to poll all but 4% of the population to get 2% accuracy.

A computational example is the problem of *polynomial identity testing.* Suppose we are given a polynomial p of degree d in n variables, not as a list of coefficients but as a formula which may not be expanded fully. How can we determine whether p is identically zero? The best known deterministic algorithm is no faster than simply expanding p out, which can take exponential time. But there is a simple polynomial-time algorithm based on the following lemma:

Lemma 1 (DeMillo-Lipton '78 [1], Schwartz '80 [2], Zippel '79 [3])**.** *If* $r_1, \dots, r_n$ *are picked randomly from* $\{1, 2, \dots, 100d\}$ *and* $p \neq 0$*, then* $\Pr[p(r_1, \dots, r_n) = 0] < 0.01$.

All we do is randomly pick such $r_1, \dots, r_n$ and compute $p(r_1, \dots, r_n)$ (which clearly takes polynomial time). If $p = 0$, we will get zero with certainty. If $p \neq 0$, then by the lemma we will get something other than zero with high probability.

Another computational example is the problem of counting the number of tilings of an n-by-n grid by dominos. This is related to the monomer-dimer problem in statistical physics, and captures various important thermodynamic properties of real systems. In general, there may be exponentially-many different tilings, and the best known deterministic algorithm for even approximately counting how many there are takes exponential time. But using randomness, there is an efficient approximating algorithm due to Luby, Randall, and Sinclair [4] (this is a *Monte Carlo* algorithm, the ideas behind which go back to von Neumann and Ulam).

Another setting where randomness can be crucial is distributed computation. Consider the *dining philosophers problem*, which is a toy model capturing the idea of resource allocation and sharing in asynchronous systems. Five philosophers sit at a round table, each with a plate of food in front of them. There are five forks placed between the plates, so that each philosopher has one fork to the left and one to the right. Each philosopher needs to eat occasionally, and requires two forks to do so. Say that the behavior of each philosopher is specified by the same program, which is run independently and concurrently by each philosopher. Is there some program which will ensure that each philosopher gets properly fed (that is, can eat an infinite number of times)? In fact, it was shown by Dijkstra that no deterministic program can achieve this — deadlock is unavoidable. However, it is possible to solve the problem with a probabilistic algorithm, which uses randomness to break the problem's symmetry (Lehmann-Rabin '81 [5]). In this case, randomness does not simply make a hard problem easy, but an impossible problem solvable.

Randomness can be fundamental in mathematical definitions, as is the case for the concept of rational behavior in game theory. For some game, a *Nash equilibrium* is an assignment of strategies to each player such that no player has an incentive to change his or her strategy given those of their opponents. Nash's fundamental theorem on such equilibria is that *every* finite game has one. However, this is only true if the players are allowed to use randomness (so called *mixed strategies*). Thus, whether or not rational participants can use randomness is of critical importance in game theory.

Example: An example [not in Widgerson's lecture] is the *matching pennies* game, where each player picks heads (H) or tails (T) and the payoff matrix is

	H	T
H	1/-1	-1/1
T	-1/1	1/-1

If each player picks H or T deterministically, there is obviously no Nash equilibrium, because whatever the outcome one player can improve his or her payoff by switching strategies. There is a mixed strategy equilibrium, however, namely when each player picks H or T with equal probability.

A final application of randomness is in cryptography and e-commerce. Here, randomness is necessary even just to define secrecy — how likely is an adversary to guess the message? The work of Shannon in information theory shows that a secret is only as good as the entropy in it. Randomness has proved indispensable in almost every part of cryptography, including public-key cryptography, digital signatures, and zero-knowledge proofs.

3 Randomness and Pseudorandomness

It is clear from these examples that randomness is useful. But where do you get random bits? Well, ask Google! Searching for "true random bits" turns up a number of companies who will happily sell them to you. Where do *they* get them from? Radioactive decay, atmospheric noise, quantum optics measurements... are these actually random? What is randomness?

3.1 Defining Randomness

Until 30 years ago, randomness was a property of events. A coin flip was random. Now, our perspective is different. Say I toss a coin, and you guess how it will land. What's the probability that you guess right? Probably you say 50%. What if I give you a laptop? Probably still 50%. Now I give you a big supercomputer. It's not 50% any more, since you can calculate the coin's trajectory and get it right every time. Randomness is in the ~~eye~~ computational power of the beholder, not in an event. As we will see, there can be great value in something which is not actually random, but "looks random" when you have limited computational ability.

3.2 Defining Pseudorandomness

Psuedorandomness is the study of deterministic structures which have some "random-like" properties (the particular properties of interest depending on the application). In computer science, we want to know how to explicitly build such structures, and how quickly this can be done.

We can make an analogy to the problem of finding explicit normal numbers. A *normal number*[1] is a real number whose digit expansion in any base b has each digit occurring $1/b$ of the time, each pair of digits occuring $1/b^2$ of the time, etc. Borel [6] proved that a random real number (with respect to Lebesgue measure) is normal, which means normality is a pseudorandom property in our terminology. The problem of building a deterministic structure with this property in this case is that of finding an explicit real number which is normal. How about π, or $\sqrt{2}$, or e? No simple example like these is known[2].

Many other problems can be phrased in this way. The P vs. NP question is an example: we know that random functions are hard to compute (require superpolynomial-size circuits) — how about TSP? The Riemann Hypothesis is another example. Consider a walk on the integers where we start at 0 and can move one unit to the left, one to the right, or stay in place at each time step. If the walk is chosen randomly, it can be shown that after N steps almost surely we will be within $O(n^{1/2+\epsilon})$ of the origin. This is our pseudorandom property, and it turns out that the Riemann Hypothesis is equivalent to the particular walk where the direction to move at time x is given by $\mu(x)$ (where μ is the Möbius function) having this property (Mertens [8]).

3.3 Coping with Imperfect Randomness

All of our examples in Section 2 assumed that we had access to perfect randomness. This is the ideal situation, but in the real world we much more frequently have *weak random sources*, which can be biased and dependent on each other. What can we do given several independent weak sources, what can we do with a single one, and what if we have no access to randomness at all? In fact, even a single weak source will suffice to make probabilistic algorithms work, and assuming P $\neq$ NP, for polynomial-time algorithms no source is needed at all!

[1]Sometimes called an *absolutely normal number*.

[2]There are some explicit normal numbers known — they are just not particularly natural [7].

3.3.1 Poor Sources: Randomness Purification

The problem with a weak random source is that it may be biased in favor of certain outcomes, and that successive trials may not be independent. The idea of *extractor theory* is to purify such sources and yield uniformly-distributed independent bits.

In the case of multiple independent weak sources, one approach to designing an extractor is that of *pseudorandom tables*. In can be shown that in a random matrix, any k-by-k "window" (formed by taking the entries at the intersections of k rows and k columns) contains $k^{1+\epsilon}$ distinct entries with high probability (for any $\epsilon > 0$). Can we generate such a table deterministically? You can try the addition and multiplication tables, for example, but in either case it is easy to find windows with only $O(k)$ distinct entries. However, we have the following theorem:

Theorem 2 (Erdös-Szemerédi '83 [9], Bourgain-Katz-Tao '04 [10])**.** *Every k-by-k window will have $\Omega(k^{1+\epsilon})$ distinct entries in* one of *the addition table or the multiplication table.*

This implies that if A, B, and C are independent weak random sources, $A+B\cdot C$ has greater entropy than any of the sources individually. This allows us combine many poor sources of randomness to create a single good source (Barak-Impagliazzo-Wigderson '04 [11]).

If we only have a single weak source (or we have multiple but they are not independent), there is still hope. Say our probabilistic algorithm requires n unbiased, independent bits. The idea is to convert a large number of "bad" bits from the source into a smaller number of better bits. For example, say that we collect n^3 bits from the source, and that they have about n^2 entropy. There are extractors which will convert these bits into sequences of n bits which are (with high probability) a perfect sample of all possible n-bit sequences. We can then feed each sequence in turn into our algorithm, and take the majority vote of the results. Because the extracted sequences are a perfect sample, with high probability we will obtain the correct result. One such extractor is due to Guruswami-Umans-Vadhan '07 [12], which can extract all the entropy from an n-bit distribution using an $O(\log n)$ truly random seed. We simply run the extractor with all the polynomially-many possible seeds, and do majority vote as above.

3.3.2 No Sources: Deterministic Derandomization

If we don't even have weak random sources, it is not clear what we can do — we can't generate entropy from nothing. The key is to remember our observation that randomness depends on computational power: if our algorithms run with limited resources, we don't need actual randomness, but merely something which is indistinguishable from randomness given those resources.

Here is the general idea. Following Goldwasser-Micali '82 [13], we say a distribution D over sequences is *pseudorandom* if for every probabilistic polynomial-time algorithm, using D instead of truly random coins gives the same output distribution. How can we deterministically generate an n-bit pseudorandom sequence? Suppose we had $k \sim c \log n$ truly random bits. A starting point would be to go from k bits of true randomness to $k+1$ bits of pseudorandomness. If we have a hard (requiring exponentially-large circuits) function f, we can do this simply by applying f to the first k bits to yield the last bit. This idea can be extended to yield an n-bit pseudorandom sequence (Nisan-Wigderson '88 [14]). Since we don't actually have a random source, as before we repeat this process for all of the $2^k \sim n^c$ seeds, feed the resulting sequences into our algorithm, and take the majority vote. This gives a deterministic[3] algorithm which runs in polynomial time: BPP = P!

The assumption in this argument is that we have a sufficiently hard function f. The best current result is the following:

Theorem 3 (Impagliazzo-Wigderson '98 [15]). *If NP requires exponential-size circuits, then BPP=P.*

We make two final notes. First, there is a partial converse to Theorem 3 (Kabanets-Impagliazzo '04 [16])! Second, it is possible to derandomize specific algorithms without assumptions. Agrawal, Kayal, and Saxena derived their primality testing algorithm [17] by designing a pseudorandom number generator good enough for a probabilistic algorithm that already existed! The same is true for Reingold's log-space algorithm [18] for undirected graph connectivity.

References

[1] Richard A. DeMillo, Richard J. Lipton, *A Probabilistic Remark on Algebraic Program Testing*, Inf. Process. Lett. (IPL) 7(4):193-195, 1978.

[2] Jack T. Schwartz, *Fast Probabilistic Algorithms for Verification of Polynomial Identities*, J. ACM 27(4):701-717, 1980.

[3] Richard E. Zippel, *Probabilistic Algorithms for Sparse Polynomials*, Lecture Notes in Computer Science 72: Symbolic and Algebraic Computation, 216-226, 1979.

[4] Michael Luby, Dana Randall, Alistair Sinclair, *Markov Chain Algorithms for Planar Lattice Structures*, SIAM J. Comput. (SIAMCOMP) 31(1):167-192, 2001.

[5] Daniel J. Lehmann, Michael O. Rabin, *On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem*, POPL 1981:133-138.

[6] Émile Borel, *Les probabilités dénombrables et leurs applications arithmétiques*, Rend. Circ. Mat. Palermo 27:247-271, 1909.

[7] Wacław Sierpiński, *Démonstration élémentaire d'un théorème de M. Borel sue les nombres absolutment normaux et détermination effective d'un tel nombre*, Bull. Soc. Math. France 45:125-144, 1917.

[8] Franz Mertens, *Über eine zahlentheoretische Funktion*, Akad. Wiss. Wein Math.-Natur. Kl. Sitzungsber. IIa 106:761-830, 1897.

[9] Paul Erdös, Endre Szemerédi, *On Sums and Products of Integers*, Studies in Pure Mathematics, 213-218. Birkhaüser, Basel, 1983.

[10] Jean Bourgain, Nets Katz, Terence Tao, *A Sum-Product Estimate in Finite Fields, and Applications*, Geom. Func. Anal. 14(1):27-57, 2004.

[11] Boaz Barak, Russell Impagliazzo, Avi Wigderson, *Extracting Randomness Using Few Independent Sources*, FOCS 2004:384-393.

[3] And because it is deterministic, it always gives the right answer!

[12] Venkatesan Guruswami, Christopher Umans, Salil P. Vadhan, *Unbalanced Expanders and Randomness Extractors from Parvaresh-Vardy Codes*, IEEE Conference on Computational Complexity 2007:96-108.

[13] Shafi Goldwasser, Silvio Micali, *Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information*, STOC 1982:365-377.

[14] Noam Nisan, Avi Wigderson, *Hardness vs. Randomness*, FOCS 1988:2-11.

[15] Russell Impagliazzo, Avi Wigderson, *Randomness vs. Time: De-Randomization under a Uniform Assumption*, FOCS 1998:734-743.

[16] Valentine Kabanets, Russell Impagliazzo, *Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds*, Computational Complexity 13(1-2):1-46, 2004.

[17] Manindra Agrawal, Neeraj Kayal, Nitin Saxena, *PRIMES is in P*, Ann. Math. 160:781-793, 2004.

[18] Omer Reingold, *Undirected ST-Connectivity in Log-Space*, STOC 2005:376-385.