Patterns hidden from simple algorithms

Madhu Sudan*

February 7, 2011

Is the number 9021960864034418159813 random? Educated opinions might vary from "No! No single string can be random", to the more contemptuous "Come on! Those are just the 714th to 733rd digits of π". Yet, to my limited mind, the string did appear random. Is there a way to use some formal mathematics to justify my naïveté? The modern theory of pseudorandomness [2, 5] indeed manages to explain such phenomena, where strings "appear" random to simple minds. The key, this theory argues, is that randomness is really in the "eyes of the beholder", or rather in the computing power of the "tester" of randomness. More things appear random to simpler, or resource-limited, algorithms than to complex, powerful, algorithms.

And why should we care? Because randomness is a key resource in computation. Monte Carlo simulations abound in the use of computation for prediction. On the theoretical side too, algorithms get simplified or speeded up incredibly if they use randomness. Fundamental tasks in distributed computing (such as "synchronization") can be solved with, and only with, randomness. And there is no hope for maintaining privacy and secrecy in the absence of randomness. At the same time much of the randomness we deal with in reality is not "pure". They don't come as a collection of independent random bits, but are generated by other processes. Knowing how an algorithm, or a computational process, works in the presence of *somewhat* random strings becomes the essence of a recurring question. (For example, should you really trust nuclear power safety calculations made by a Monte-Carlo algorithm using randomness from the C++ `rand` program?)

Questions such as the above get formalized in the theory of pseudo-randomness as follows: It considers some source of randomness X (formally given by some probability distribution over n-bit strings), and a class of "Boolean" algorithms $\mathcal{A}$ (algorithms that decide Boolean questions) and asks if some algorithm in $\mathcal{A}$ behaves very differently given access to a random string generated by X, than it does on pure random strings? If every algorithm in $\mathcal{A}$ behaves roughly the same on X as on pure randomness, we say X is *pseudo-random* to $\mathcal{A}$. In the example above, $X = X_\pi$ may be the source that picks a random integer i between 1 and 10000 and outputs $\pi[i+1], \ldots, \pi[i+20]$ where $\pi[j]$ denotes the jth digit of π. Given some class of algorithms $\mathcal{A}$ one could now ask, does X_π look pseudo-random to $\mathcal{A}$?

Unfortunately answering such questions leads to a fundamental challenge in the theory of computing. Proving, say, that X_π looks random to $\mathcal{A}$ involves showing that no algorithm in $\mathcal{A}$ can *detect* patterns that the digits of π show. And proving that some class of algorithms can't do some task is a major challenge to theoretical computer science—with "Is P = NP?" question being the most notorious example.

*Madhu Sudan (`madhu@mit.edu`) is a Principal Researcher at Microsoft Research New England and the Fujitsu Professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology.

Given such major obstacles to analyzing the seeming randomness in strings, it is no surprise that the study of pseudo-randomness remains in its infancy. The research highlight article by Braverman sheds new light on this field. It shows that a broad class of sources, as long as they have sufficient "local randomness", fool a broad, but relatively simple class of algorithms, those that compute "AC0 functions". AC0 functions are those whose input-output behavior can be described by a Boolean logic circuit consisting n input wires and polynomially many NOT gates and AND and OR gates of arbitrary fan-in. AC0 functions correspond to tasks that can be computed in constant time on highly parallel machines. These functions are at the forefront of the class of computational tasks that we do seem to understand. For instance, AC0 functions can compute the sum of two $n/2$ bit integers, but not their product (and so we do know things they can't do). In fact, we even knew a explicit source of relatively small entropy that appeared pseudo-random to this class [4]. Yet our understanding of the essence of what sources fooled this class of algorithms was unknown, leading Linial and Nisan [3] to conjecture that a certain form of "local randomness" was sufficient to seem random to this class of algorithms. The "local randomness" they pointed to is widely termed "limited independence". A source X is said to be *k-wise independent* if any particular k bits of the random source are totally random and independent. Linial and Nisan conjectured that for every constant depth circuit, some $(\log n)^{O(1)}$-wise independence would look like pure n bits of randomness. For over two decades this conjecture remained unresolved. Only recently, Bazzi [1] resolved a special case of the conjecture (for the case of AC0 functions corresponding to depth 2 circuits, or "DNF formulae").

Now Braverman's work resolves the conjecture affirmatively. In the process he reveals novel, elegant, ways in which AC0 functions can be described by low-degree multivariate polynomials, showing some of the interplay between Boolean computation and more classical mathematics. We need to see many more such connections before we can hope to address the wide challenges of computational complexity (and P vs. NP). Indeed even to address the question implied by the opening paragraph "Are the digits of π pseudo-random to AC0 functions?", one needs to understand much more. Fortunately, the work of Braverman may have reduced this question to a purely number-theoretic one: Are the digits of π locally random?

References

[1] Louay M. J. Bazzi. Polylogarithmic independence can fool dnf formulas. *SIAM J. Comput.*, 38(6):2220–2272, 2009.

[2] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[3] Nathan Linial and Noam Nisan. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365, 1990.

[4] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[5] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE, 1982.