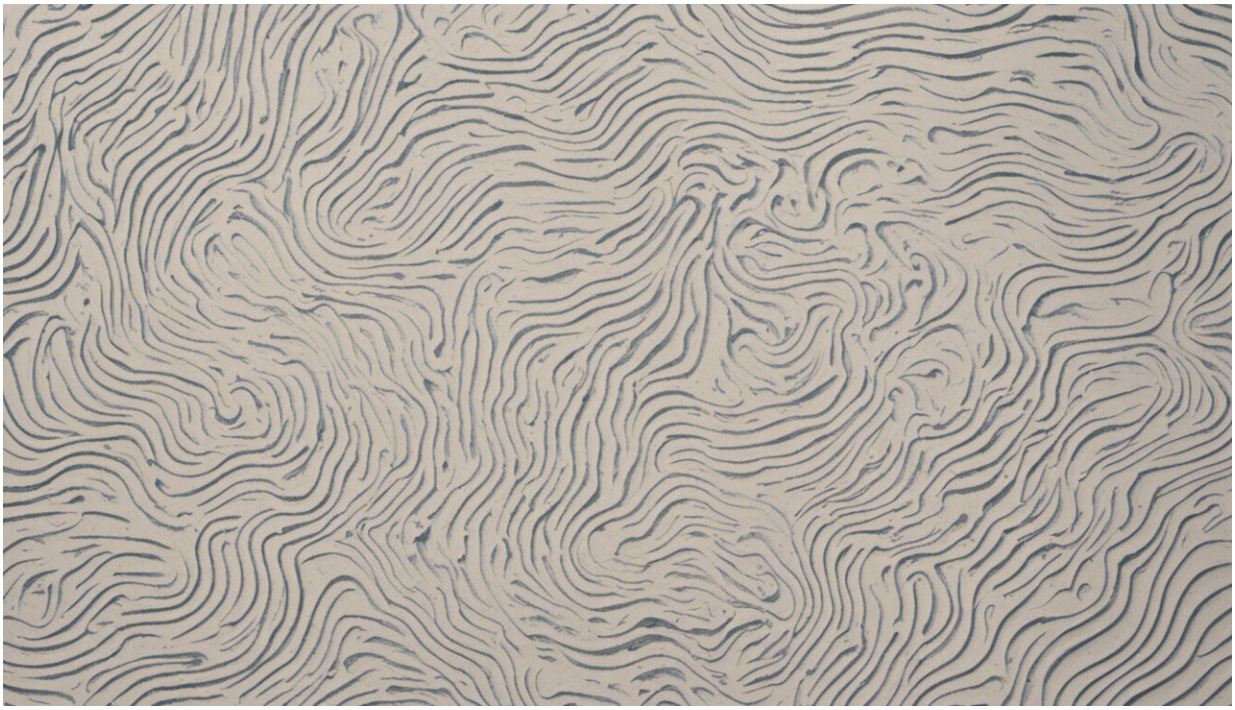
How random is your randomness, and why does it matter?

September 19 2016, by David Zuckerman And Eshan Chattopadhyay



Credit: AI-generated image (disclaimer)

Randomness is powerful. Think about a presidential poll: A random sample of just 400 people in the United States can accurately estimate Clinton's and Trump's support to within 5 percent (with 95 percent certainty), despite the U.S. population exceeding 300 million. That's just one of many uses.

Randomness is vital for computer security, making possible secure encryption that allows people to communicate secretly even if an adversary sees all coded messages. Surprisingly, it even allows security to be maintained if the adversary also knows the key used to the encode the messages.

Often random numbers can be used to speed up algorithms. For example, the fastest way we know to test whether a particular number is prime involves choosing random numbers. That can be helpful in math, computer science and cryptography, among other disciplines.

Random numbers are also crucial to simulating very complex systems. When dealing with the climate or the economy, for example, so many factors interact in so many ways that the equations involve millions of variables. Today's computers are not powerful enough to handle all these unknowns. Modeling this complexity with random numbers simplifies the calculations, and still results in accurate simulations.

But it turns out some – even most – computer-generated "random" numbers aren't actually random. They can follow subtle patterns that can be observed over long periods of time, or over many instances of generating random numbers. For example, a simple random number generator could be built by timing the intervals between a user's keystrokes. But the results would not really be random, because there are correlations and patterns in these timings, especially when looking at a large number of them.

Using this sort of output – numbers that appear at first glance to be unrelated but which really follow a hidden pattern – can weaken polls' accuracy and communication secrecy, and render those simulations useless. How can we obtain high-quality randomness, and what does this even mean?

Randomness quality

To be most effective, we want numbers that are very close to random. Suppose a pollster wants to pick a random congressional district. As there are 435 districts, each district should have one chance in 435 of being picked. No district should be significantly more or less likely to be chosen.

Low-quality randomness is an even bigger concern for computer security. Hackers often exploit situations where a supposedly random string isn't all that random, like when an encryption key is generated with keystroke intervals.

It turns out to be very hard for computers to generate truly random numbers, because computers are just machines that follow fixed instructions. One approach has been to use a physical phenomenon a computer can monitor, such as radioactive decay of a material or atmospheric noise. These are intrinsically unpredictable and therefore hard for a potential attacker to guess. However, these methods are typically too slow to supply enough random numbers for all the needs computers and people have.

There are other, more easily accessible sources of near-randomness, such as those keystroke intervals or monitoring computer processors' activity. However, these produce random numbers that do follow some patterns, and at best contain only some amount of uncertainty. These are low-quality random sources. They're not very useful on their own.

What we need is called a randomness extractor: an algorithm that takes as input two (or more) independent, low-quality random sources and outputs a truly random string (or a string extremely close to random).

Constructing a randomness extractor

Mathematically, it is impossible to extract randomness from just one low-quality source. A clever (but by now standard) argument from probability shows that it's possible to create a two-source extractor algorithm to generate a random number. But that proof doesn't tell us how to make one, nor guarantee that an efficient algorithm exists.

Until our recent work, the only known efficient two-source extractors required that at least one of the random sources actually had moderately high quality. We recently developed an efficient two-source extractor algorithm that works even if both sources have very low quality.

Our algorithm for the two-source extractor has two parts. The first part uses a cryptographic method called a "nonmalleable extractor" to convert the two independent sources into one series of coin flips. This allows us to reduce the two-source extractor problem to solving the a quite different problem.

Suppose a group of people want to collectively make an unbiased random choice, say among two possible choices. The catch is that some unknown subgroup of these people have their heart set on one result or the other, and want to influence the decision to go their way. How can we prevent this from happening, and ensure the ultimate result is as random as possible?

The simplest method is to just flip a coin, right? But then the person who does the flipping will just call out the result he wants. If we have everyone flip a coin, the dishonest players can cheat by waiting until the honest players announce their coin flips.

A middling solution is to let everyone flip a coin, and go with the outcome of a majority of coin flippers. This is effective if the number of

cheaters is not too large; among the honest players, the number of heads is likely to differ from the number of tails by a significant amount. If the number of cheaters is smaller, then they won't be able to affect the outcome.

Protecting against cheaters

We constructed an algorithm, called a "resilient function," that tolerates a much larger number of cheaters. It depends on more than just the numbers of heads and tails. A building block of our function is called the "tribes function," which we can explain as follows.

Suppose there are 44 people involved in collectively flipping a coin, some of whom may be cheaters. To make the collective coin flip close to fair, divide them into 11 subgroups of four people each. Each subgroup will call out "heads" if all of its members flip heads; otherwise it will say "tails." The tribes function outputs "heads" if any subgroup says "heads;" otherwise it outputs "tails."

The tribes function works well if there is just one cheater. This is because if some other member of the cheater's subgroup flips tails, then the cheater's coin flip doesn't affect the outcome. However, it works poorly if there are four cheaters, and if those players all belong to the same subgroup. For then all of them could output "heads," and force the tribes function to output "heads."

To handle many cheaters, we build upon work of Miklos Ajtai and Nati Linial and use many different divisions into subgroups. This gives many different tribes functions. We then output "heads" if all these tribe functions output "heads"; otherwise we output "tails." Even a large number of cheaters is unlikely to be able to control the output, ensuring the result is, in fact, very random.

Our extractor outputs just one almost random bit – "heads" or "tails." Shortly afterwards Xin Li showed how to use our algorithm to output many bits. While we gave an exponential improvement, other researchers have further improved our work, and we are now very close to optimal.

Our finding is truly just one piece of a many-faceted puzzle. It also advances an important field in the mathematical community, called Ramsey theory, which seeks to find structure even in random-looking objects.

This article was originally published on The Conversation. *Read the* original article.

Source: The Conversation

Citation: How random is your randomness, and why does it matter? (2016, September 19) retrieved 27 April 2024 from https://phys.org/news/2016-09-random-randomness.html