Randomness + Determinism = Progresses
Why Random Processes Could Be Favored by Evolution

Nicolas Schabanel

CNRS, Université Paris Diderot (LIAFA UMR 7089)
Case 7014, 175 rue du Chevaleret, 75205 Paris, France.
http://www.liafa.univ-paris-diderot.fr/∼nschaban.

Université de Lyon (IXXI), École normale supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.

Abstract. Biologists are somehow pioneers on the idea that progress can be driven by randomness: randomness is one of the main engine of evolution; small variations induced by randomness coupled with natural selection allows the species to self-adapt to their moving environment. Studies from the last 40 years in computer science suggest that randomness is in fact able of doing much more and revealed unexpected possibilities which might appear impossible at first. Furthermore, it turns out that these discoveries are faster, cheaper and above all exponentially thriftier than their deterministic alternatives. This means that random explorations would almost surely generate a stochastic process way before any equivalent deterministic counterpart is found. It follows that most likely these processes are favored by evolution and should thus be known to anyone dealing with systems (alive or not) having access to random sources. This article presents some of these counter-intuitive results as a possible source of inspiration for studying systems fed with randomness.

Keywords. Randomness, Model, Algorithms, Applications of Computer Science to Biology Modeling.

1. Introduction

Since the 1970s, with the work of the two pioneers Erdös and Rabin [12, 15, 5, 1], several important progresses in computer science have shown how intuition can fail to capture the true power of randomness. Randomness is often associated with the notion of noise, unpredictability, regularization (law of large numbers), inhomogeneity... It then appears as arbitrary, independent of our will and beyond our understanding, a source that perturbs a system: sometimes for the good (inhomogeneous gas in space leads to the formation of stars and planets; or it allows to extract simple laws from a very complicated situation in ideal gas) sometimes for the bad (noise over communication channels).

Darwin and later on Wallace were among the first to envisage randomness as the engine or the fuel of a system rather than a perturbation: the *random changes* from parent to children *coupled* with some *deterministic selection* process (the necessary adaption to their moving environments,...) yields to the evolution of species, a phenomenon that could not take place if one of these two ingredients (random changes and deterministic selection) was missing: pure randomness is unable to focus on efficient moves, and deterministic process alone would lack the necessary creativity. Another well-known example where randomness plays a key role is game theory: it is known since von Neumann [17, 11] that optimal strategies are probabilistic for zero-sum games (i.e. games where the loser pays to the winner). For instance, in scissors-paper-stone, the optimal strategy consists in playing scissors, paper or stone uniformly at random. Any deterministic strategy can be easily fooled. Von Neumann's result implies that some randomness has to be input into key political decisions in order to make the "right" choice according to the optimal probabilistic strategy. Ruelle [14] sees in this result an interesting explanation for the emergence of religions: their sacrificial rituals are a very reliable source of randomness and civilizations using these rituals in their decision process could thus have had a decisive advantage to take over civilizations making their decision deterministically.

Researches from the past two decades in computer science pushed these principles much further and have taught us that instead of being confronted, randomness could be used to design better and more efficient systems. In a series of examples, we will see in the next sections how to count up to 10^{309} on our ten fingers only; how to correct a system with unknown (deterministic) bugs by calling it several times on random entries and recombining the outputs; and most strikingly, how to convince someone that we know something without revealing *anything* about it (the Graal in system security), showing that the true nature of information is quite puzzling.

Alongside to these results in computer science, the emerging field of compressive sensing reveal similar surprising trend reversals in physics. A wise use of randomness allows for instance much better signal sampling procedures: when the sample dates are chosen at random, the number of samples required to reconstruct a given signal does not depend anymore on the highest frequency in the signal but on the number of frequencies in the signals which is several order less in practice, allowing to reduce the time spent by pa-

tients in MRI machines for instance (e.g., [4]). Interestingly enough, these results heavily rely on the digitalization of the data. This digitalization allows to capture the data non-figuratively (for instance a superposition of the very same image translated in random directions) and then to process them by sophisticated mathematical transformations to yield the desired result. An other example is the case of low-light photography: a wide open diaphragm would reduce considerably the depth of field whereas randomly placed pin-holes yield a superposition of various infinite-depth-of-field captures of the scene on the digital sensor. This superposition can be easily processed to obtain a perfect image with infinite depth-of-field and with the same amount of light as the wide open diaphragm. The same kind of ideas allows to annihilate the motion blur in low-speed photography (e.g., [13]).

The power of randomness comes from essentially two key properties: 1) it is *unpredictable* and thus it prevents from been fooled by a noxious agent which tries to bypass the protections; 2) it is able to *select uniformly* a random individual and thus to scan reliably the *whole space of possibilities*, even if this space is totally unstructured. The combination of these two properties allows to accomplish astonishing results that are detailed in the following.

Mathematical Analysis of the Stochastic Counter

Let X_n denote the random variable for the value of the stochastic counter X after n increments. By definition:

$$X_0 = 0$$
$$X_{n\geqslant 1} = \begin{cases} 1 + X_{n-1} & \text{with probability } 2^{-X_{n-1}} \\ X_{n-1} & \text{otherwise} \end{cases}$$

Given that X_{n-1} is x, the expected value of 2^{X_n} is:

$$\begin{aligned}\mathbb{E}\big[2^{X_n}\big|X_{n-1}=x\big] &= 2^{-x}\cdot 2^{x+1} + (1-2^{-x})\cdot 2^x \\ &= 2^{X_{n-1}} + 1\end{aligned}$$

By immediate recurrence, since $2^{X_0} - 1 = 0$, the expected value output after n increments, $2^{X_n} - 1$, is n, as claimed:

$$\boxed{\mathbb{E}\big[2^{X_n} - 1\big] = \mathbb{E}\big[2^{X_{n-1}} - 1\big] + 1 = (n-1) + 1 = n}$$

Figure 1: Analysis of the stochastic counter.

2. Too small is big enough

Let us take a first example: how much memory do we need to count up to n? Here is the setting: imagine that we want to keep track inside each internet router of the number of times a specific event occurs, for instance, the number of packets passing through the router towards a specific direction in order to detect possible attacks as soon as possible; how much resources does it require to count n events one-by-one? Deterministically, we need to write down n which requires a *logarithmic* number of digits, i.e. $\log_2 n$ bits in binary or $\log_{10}(n)$ digits in decimal: counting ten billions (10^{10}) packets would require $\log_{10}(10^{10}) = 10$ digits in decimal or $\log_2(10^{10}) \doteq 34$ bits in binary. One can save a lot of resources using instead a stochastic counter X as follows [10, 6]: start with $X = 0$ and each time you see an event then you increase X by one with probability $1/2^X$ and leave it unchanged otherwise; if asked for the number of events, output $N = 2^X - 1$. It can easily be shown by recurrence that the expected value of N is precisely n the number of events! (see Fig. 1 for a proof, and we refer to [6] for improvements on the variance). Now X requires logarithmically less resources then n: only $\log_2(\log_2(n))$ bits in binary or $\log_{10}(\log_2(n))$ in decimal, that is to say 6 bits instead of 34 in binary or only 2 digits instead of 10 in decimals for $n = 10^{10}$ (ten billions of) events! This simple stochastic counter allows then to save 83% of the resources for $n = 10^{10}$ without compromising the accuracy of count on average, and it saves more and more as n grows, more than 99% of the resources for very large numbers as $n = 10^{309}$ by using 10 bits only instead of 1024! As illustrated in Fig. 4.

Why does this saving matter? Because it is *exponential*: using a deterministic counter requires exponentially more resources than the probabilistic counter. Concretely, imagine that the counter is realized by network of AND/OR gates (natural neurons can implement this type of gates). A deterministic counter would require as many gates as *all the possible networks of gates altogether* of the size of the one performing a probabilistic counter (see Fig. 2). This implies, if one adopts an evolutionary point of view: nature would spend less resources scanning randomly all the possible gate networks of the same size as the one performing a probabilistic counter than trying only *one single* network of the size required to perform one single deterministic counter (see Fig. 3). This means that nature would almost surely find and be satisfied of the probabilistic counter network long before it even tries any deterministic ones. Furthermore, the probabilistic counter is much more resources efficient to run than the deterministic one: it is thus much easier to find by random tries and almost much more economical to run. The deterministic gate network would thus almost surely be condemned in a natural selection scenario with reliable sources of

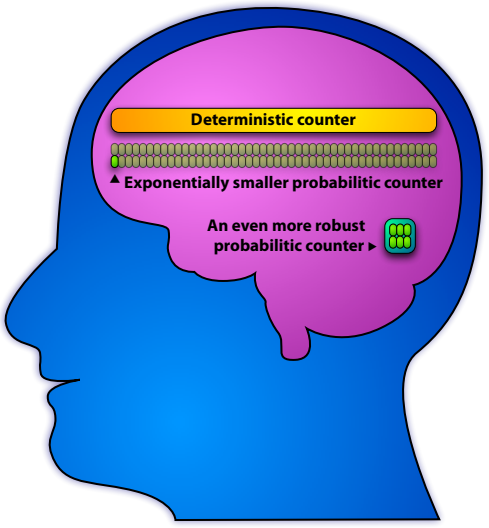


Figure 2: Probabilistic counters save a lot of resources with respect to deterministic counters. Replicating them to reduce their error probability is still much more economical and frees space to do other things.

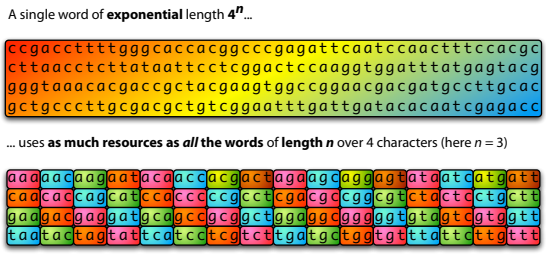


Figure 3: A single word of exponential length consumes as much resources (characters) as *all* words of exponentially smaller length — as an illustration for one single deterministic (exponentially larger) neuronal network consuming the same number of neurons as required to build all possible stochastic neuronal networks of the same size as the neuronal network performing the stochastic counter.

randomness.

Reliable probabilistic counters save resources on the long run. Note also as shown on Fig. 2, that represented as a gate network, the probabilistic counter uses exponentially less space in the brain as the deterministic one. One can thus reduce the probability of error by replicating this counter and it still costs much less than the deterministic counterpart. Replicating it until obtaining perfect counting would still use no more resources as deterministic counters (see [6]).

3. Information conveyed in random signals

Consider now the scenario of an unreliable multiplier unit: it only responds correctly to a fraction $1-\epsilon$ of the possible inputs. For instance, $\epsilon \leqslant 10\%$ if its outputs are correct for at least 90% of the inputs. This means that for at least 90% of the pairs of integers (a, b), its output is indeed the product $a \times b$ whereas it can be an arbitrary integer for any of the remaining 10% of pairs. The question is: can we still compute the right value for the product of two integers a and b for which the unit always output the wrong answer? I.e., can we correct a buggy multiplier using only the buggy multiplier? The answer is Yes, and it uses randomness (and a reliable adder).

Buggy systems can be made reliable as follows. The self-correction scheme has been proposed by Blum, Luby and Rubinfeld [2, 18] and is illustrated in Fig. 5. The key is to use the property of the multiplication that you can write the product of any two integers as the sum of four products of uniformly random pairs of integers. Assume we want to compute the product $a \times b$ of two integers a and b with a buggy multiplier. We first draw to random numbers r and s uniformly and independently.[1] We then compute the sums $a+r$ and $b+s$ (we assume that only the multiplier is buggy and that we can compute every addition correctly, this is not a restrictive assumption since reliable addition can be obtained from a buggy adder by similar technics, see [18]). Then, we ask the multiplier for its outputs for the four products $m_1 = (a+r) \times (b+s)$, $m_2 = (a+r) \times s$, $m_3 = r \times (b+s)$ and $m_4 = r \times s$. We then output $m = m_1 - m_2 - m_3 + m_4$.

What is the probability that the output is correct, i.e. that $m = a \times b$?. First, remark that if all four products m_1, m_2, m_3, and m_4 are correct then m is indeed the right value:

$$\begin{aligned} m = m_1 - m_2 - m_3 + m_4 = \; & (a+r) \times (b+s) \\ & - (a+r) \times s \\ & - r \times (b+s) + r \times s \\ = \; & a \times b \end{aligned}$$

Now, what is the probability that all four outputs m_1, m_2, m_3, and m_4 are correct? Because s and r are independent uniform random numbers, each of these products is the product of two independent uniform random numbers (the sum of a uniform random number with an other number is still a uniform random number, see[1]). The buggy multiplier will thus output the right answer for each of them with probability at

[1] In order to be rigourous, we would need to conduct every operation modulo some large integer, see [18].

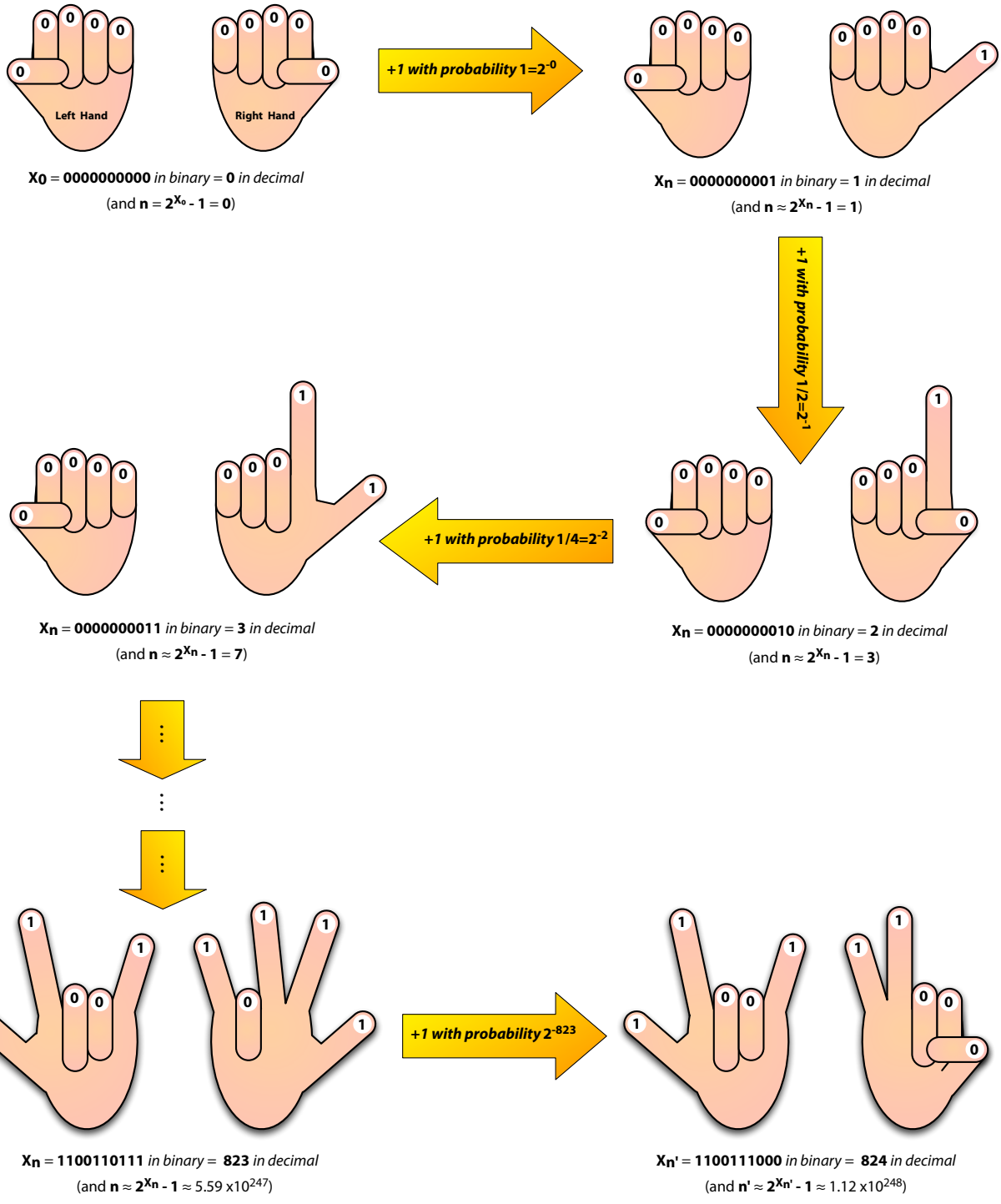


Figure 4: **Counting up to 10^{309} on our ten fingers only.** At any time the expected value of $(2^{X_n} - 1)$ is precisely n after n increments. Each of the ten fingers store a bit of X_n: if the i-th finger from the right is straight, the i-th bit of X_n writen in binary is 1, and 0 otherwise. Our ten fingers allows to increase X up to $2^10 - 1 = 1023$ and thus to count up to $n = 2^{1023} - 1 \sim 10^{309}$. Note that given X, increasing X by one with probability 2^{-X} is easily done with ten fingers: we increase X by one if we obtain X "head"s in a raw by flipping X times independently an unbiased coin.

least $1-\epsilon$ ($= 90\%$), since each input, taken independently, is uniformly random. It follows that all four outputs will be the correct products with probability at least $1-4\epsilon$ ($\geqslant 60\%$) by the union bound (even if they are correlated). If we are not satisfied with this probability, just rerun it several times, take the majority and the probability of error will drop exponentially fast to a tiny pourcentage.

Randomness allows to correct an unreliable systems with unknown bugs. Surprisingly, in this scheme, we do not need to know what is wrong in the buggy multiplier to obtain a correct answer from it with high enough probability for all inputs. We do not even need to known for which entries the output was incorrect. We just need to know that errors do not occur for too many entries. This kind of scheme applies to a wide variety of functions [18] including addition, Fourrier transforms, linear operations,... It also suggests that if some living form contains some source of randomness (as many do), then for at least some specific tasks, the unit supposed to accomplish this task does not need to perform it perfectly. Some error may occur that can be corrected with this kind of scheme. Furthermore, if building a buggy unit costs much less resources, then from an evolutionary point of view, this buggy unit is much more likely to arise in nature.

Random signals may convey deterministic informations. An other important remark is that whatever a and b are, the "signals" (the integers) exchanged with the buggy unit (the multiplier) are pure random numbers (see Fig. 5) and apparently do not convey any information at all. If the evolutionary hypothesis above is correct, it may be the case in nature that some apparently unstructured, random signals convey in fact a deterministic information, once recombined together; the randomness being used just to improve the robustness of the system.

4. Convincing without revealing anything

This last section presents one of the most puzzling results of computer science from the last decades: the *zero-knowledge proofs* introduced by Goldwasser, Micali and Rackoff [7]. This domain started from the observation that before delivering money to someone, an ATM[2] needs to make sure that he is the right owner of the bank account and this is done by providing to the ATM at the same time your credit card and the corresponding pin code which is assumed to be only known by you. If the ATM has both, it then assumes that you are indeed who you claim to be. This protocol has however an important flaw: at some point, the ATM has both your card and your pin code and could make a copy of both and steal your identity (this technics is known as *scamming* and is not unusual in the USA). The question raised by Goldwasser *et al* is whether one could just *convince* the ATM that the client knows the pin code *without revealing anything* about it: the ATM would be convinced that the client is who he pretends to be, and the ATM would not learn anything about the pin code and thus could not steal his identity.

Zero-knowledge principle. In the classic ATM withdrawal scenario, you are identified as the rightful owner of the account by your ability to provide the pin code matching the credit card, which is hard to guess by any other person since he has only $1/10,000$ probability to guess it correctly and can try only three times (which yields a probability 0.3% for successfully guessing the pin code with no more than three trials). In zero-knowledge proof, you are identified by your ability to solve one *hard* problem, that is to say a problem that no one else can solve. You can solve this problem because it was constructed especially for you together with its solution (as the credit card with its built-in pin code), and it is hard because if someone does not have the solution, then constructing it from the problem is impossible in reasonable time.[3] The key to zero-knowledge protocols is to find a problem for which we can test if you know the solution without needing to know anything about how to solve the problem. Sounds impossible? Well, here is how to accomplish this paradoxical task. Without surprise, it makes extensive use of randomness.

The leaves-counting example. Zero-knowledge protocols are based on known-to-be-hard problems in computer science, such as *graph isomorphism* that will be discussed later on. But let us rather focus on a toy example to understand how zero-knowledge proofs are indeed possible. Suppose that we want to identify Alice and that Alice is known to be the only one who knows how to *count the leaves of real-life trees*, this is her secret, no one else knows how to solve this very hard problem in reasonable time, but Alice knows and does not want to leak any part of her secret. How an ATM can be convinced that Alice knows this secret without knowing how to do it itself? The protocol is illustrated in Fig. 6.

First the ATM chooses a random tree around and ask Alice for its number of leaves. Alice answers some number, 123,470 for instance. Of course, the

[2] *ATM: Automated Teller Machine* used to withdraw cash with a credit card.

[3] The NP-complete problems in computer science are precisely hard in that sense.

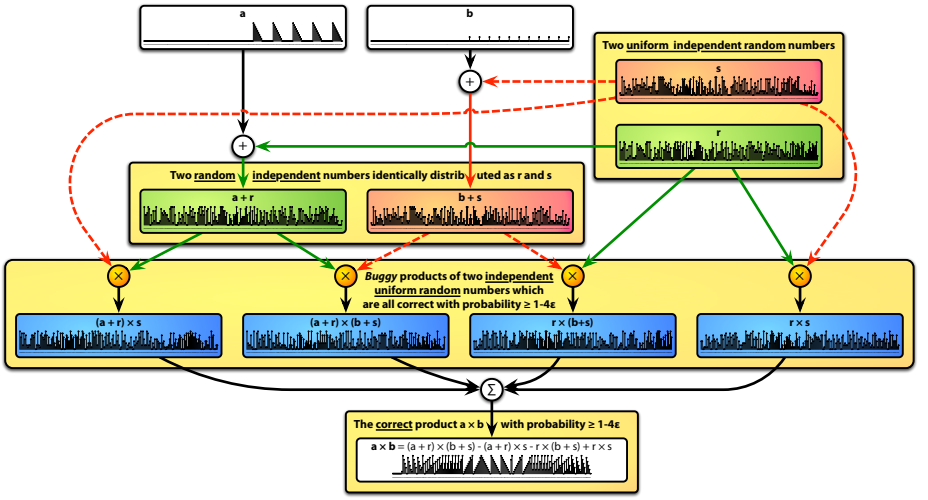


Figure 5: **The self-corrected version of a buggy multiplier.** Here we want to multiply two 200-digits numbers a and b and for that purpose we use two 200-digits uniform and independent random numbers r and s. Each number is represented as an abacus (the i-th digit from the right of each number is represented by a bar and a ball whose height is the value of the corresponding digit). As one can observe in the figure, a, b and $a \times b$ do not look random, whereas all the other numbers "look" perfectly random (and they are). The scheme to compute $a \times b$ is to ask the buggy multiplier to do the four multiplication $(a + r) \times (b + s)$, $(a + r) \times s$, $r \times (b + s)$ and $r \times s$ and to recombine the four outputs to obtain the desired product. In the graphics above, the random numbers that depends on r ($a + r$ and r) and on s ($b + s$ and s) lie on path colored in green and red respectively and it can thus clearly be seen that each of the four multiplications is conducted only on truly independent uniform pairs of random numbers (a red and a green) and has thus each a probability at most ϵ to fail. As one can see the recombination of the four random-looking intermediate products yields the expected non-random looking product $a \times b$. Random looking signals may thus convey, once recombined, deterministic information.

ATM cannot check whether this is right or wrong. Then, the ATM blindfolds Alice, draws a small random number, for instance 41, and removes 41 leaves from the tree. The ATM then asks Alice to unfold her eyes and asks her for the number of leaves again. If her answer is different from 123,429 = 123,470−41, the ATM knows *for sure* that she does not know how to count the leaves of trees and that she cannot be Alice. Now, if Alice's answer is 123,429 then she correctly guessed the number of leaves removed and the only way for her to guess it for sure was to know how to count the leaves. The ATM is thus convinced that Alice probably knows how to count the leaves of trees. If not, the ATM can redo the test several times, the error probability will drop exponentially fast and be soon much lower than the 0.3% of the current credit card protocol.

What did the ATM learn? Nothing besides only being convinced that Alice knows the secret. Not even the number of leaves in the tree it choosed! Indeed, Alice concealed this information by drawing first a small random number, for instance 14, and then adding it to the real number of leaves: she counted 123,456 leaves, but answered 123,456 + 14 = 123,470. For the second question, Alice counted 123,415 leaves and answered 123,415 + 14 = 123,429 leaves. The ATM did not learn anything from this protocol besides the conviction that Alice knows how to count the leaves of trees.

How does it work?. The essential ingredient here was the unpredictability of random numbers: Alice protected her secret by adding an unpredictable number to her counts, and the ATM ensured the validity of its test by removing an unpredictable number of leaves. This example shows how deep the notion of information is and how wrong our intuition can be: there is no absolute necessity to know or learn anything about something, to certify that someone knows it.

Zero-knowledge in real-life. A typical problem that is used in zero-knowledge protocols is the problem of *Graph Isomorphism*. A graph is a set of points connected by a set of straight lines (called edges). The problem of Graph Isomorphism consists in deciding whether one graph can be obtained from another by moving its points. It turns out to be a very hard problem that no one knows presently how to solve in practice (see Fig. 7). Instances of this problem are however very easy to build: just draw a random graph and move its points randomly to obtain an isomorphic copy of it; the credit card will be the pair of graphs and the secret pin code will be the secret movements of the points. It turns out that one can test if someone knows the secret without leaking any information on the movement of the points. Note also that zero-knowledge proofs are bounded to number problems (e.g., graph isomorphism), and that not all problems admit zero-knowledge proofs but this would take us too far.

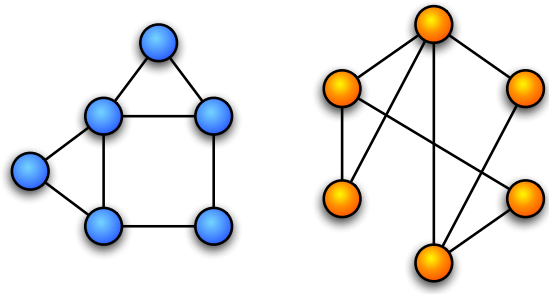


Figure 7: Are these two graphs isomorphic?

5. Conclusion

Randomness is much more than just noise when properly coupled with deterministic decisions. Its ability to draw uniformly over a large set of possibilities allows to bypass unknown bugs in programs (such as for the multiplier), count correctly on average without keeping track of the precise count (as in the probabilistic counters). Its unpredictability allows to be convinced that someone knows a secret without any leak of his secret. Invisible correlations between random-looking signals may convey deterministic information once combined together. Furthermore the stochastic systems obtained by this way are most of time considerably more resource-efficient than their more familiar deterministic counterparts. Smaller, they could be more likely to appear randomly in nature. More resource-efficient, they provide a strong advantage in the long run. Counter-intuitive, they need to be learned in order to avoid been puzzled when encountered for the first time. Moreover, their failing error is most of time exponentially small and thus their failure would occur only after an exponential time which is at the living forms time scale close to eternity. Such an unit should thus be considered perfectly reliable at our time scale.

Could natural systems be random?. One can furthermore remark that the "structure" of engineered systems seems to differ considerably from natural systems (see for instance Fig. 8): the "structure" of natural systems is often much less visible with complex interactions between entities (there could even be no structure at all!), as opposed to engineered systems which often adopt hierarchical structures because it is suitable for proving the correctness of their behavior and thus providing the required level of certification to avoid accidents. These hierarchical structure

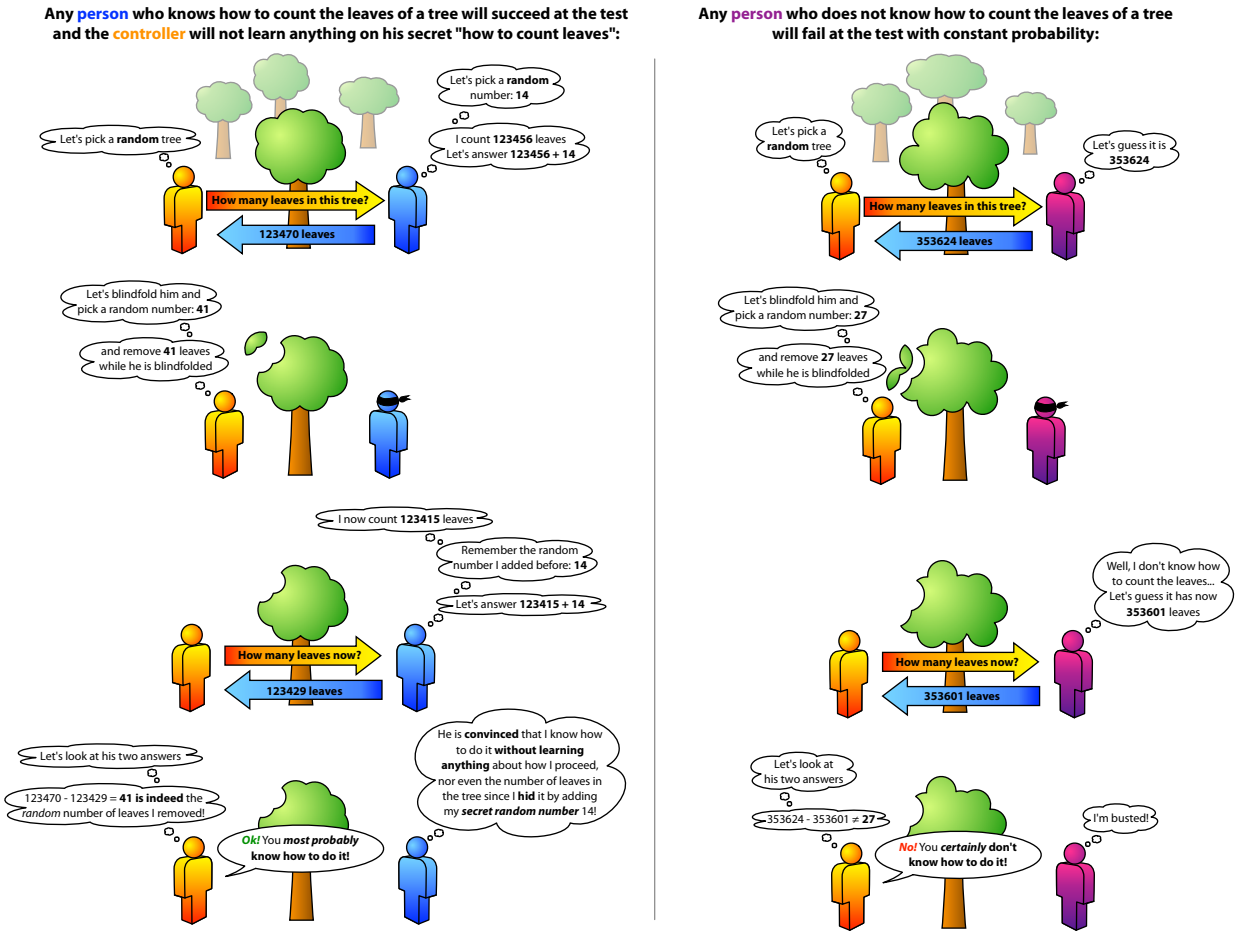
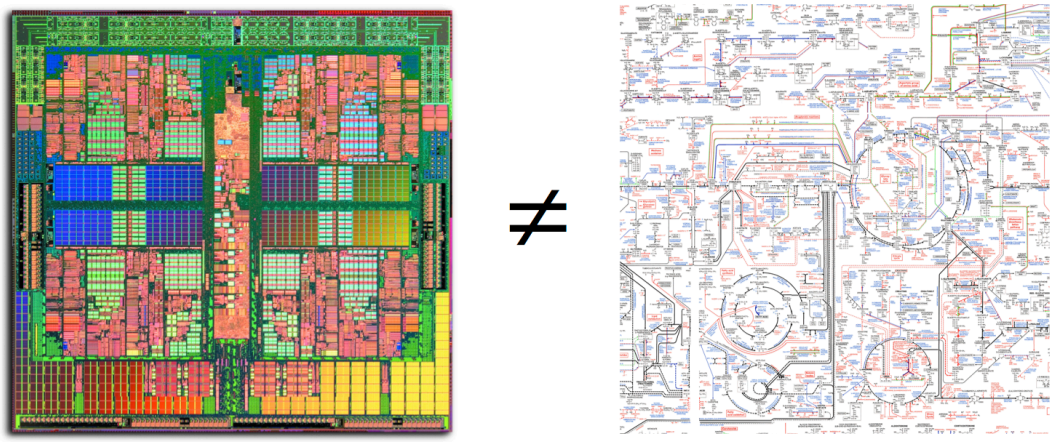


Figure 6: The zero-knowledge protocol for leaves-counting.



Engineered systems
(Quadcore Opteron processor)

≠

Natural systems
(Extract of metabolic pathways)

Figure 8: Engineered vs natural organizations.

can be exponentially less resource efficient, e.g. the counter case. Natural systems just need to work (no proof asked). Furthermore, they may have been obtained by a succession of "lucky" random changes, possibly favoring the solutions with moderate impact of their environment. Adopting a computer scientist point of view, the impact on the environment can be minimized by favoring the smaller "programs/units" accomplishing a given task: it requires less molecules to built and consumes less energy to run. As argued earlier: smaller programs not only save resources but may also be more likely to be found by random searches such as evolution theory suggests (this statement would need to be argued more precisely and should be the object of new developments that would unfortunately not fit in the format of this article). If smaller "programs/units" were to be favored in nature, this could imply other interesting connections between natural systems and randomness. Indeed, minimal programs in computer science are known to have very particular forms that look totally random and present no evident structure: formally speaking, they have a very high Kolmogorov complexity, but being of very high Kolmogorov complexity is just an other definition of being random (see the work of Solomonoff, Komogorov, Chaitin, and Martin-Löf [16, 8, 3, 9]). Informally, if the program had a structure, then we could use its structure to compress it a little more and make it smaller: it follows that minimal programs are necessarily random-looking. Thus, the fact that smaller solutions could be favored in the evolution process may explain why natural systems seem to "look random": they would indeed be close to random. There could thus be new intriguing links to discover between natural patterns and randomness following this direction (experimentally and theoretically) which might lead to a better understanding of the consequences of natural selection on the evolution of natural systems.

References

[1] Noga Alon and Joel Spencer. *The Probabilistic Method (2nd ed.)*. Wiley-Interscience, 2000.

[2] Manuel Blum, Michael Luby, and Ronnit Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proc. of ACM Symp. on Theory of Computing (STOC)*, pages 73–83, 1990.

[3] G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[4] D. L. Donoho. Compressed sensing. *IEEE Trans. on Information Theory*, 52(4):1289–1306, 2006.

[5] Paul Erdős. Graph theory and probability. *Canad. J. Math*, 11(0):34–38, 1959.

[6] Philippe Flajolet. Counting by coin tossing (keynote paper). In *LNCS Proc. of Asian Computing Science Conference (ASIAN)*, volume 3321, pages 1–12, 2004.

[7] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge compexity of interactive proof systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[8] Andrei N. Kolmogorov. Trois approches à la définition du concept de quantité d'information. *Probl. Peredachi Inform.*, 1:3–11, 1965.

[9] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

[10] R. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1977.

[11] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge university press, 1995.

[12] Michael O. Rabin. Probabilistic algorithms. In J.F. Traub, editor, *Algorithms and Complexity, Recent Results and New directions*, pages 21–39. Academic Press, New York, 1976.

[13] Ramesh Raskar and Jack Tumblin. *Computational Photography: Mastering New Techniques for Lenses, Lighting, and Sensors*. A K Peters/CRC Press, 2012.

[14] David Ruelle. *Hasard et Chaos*. Odile Jacob, 1991.

[15] Dennis Shasha. An interview with michael o. rabin. *Communications of the ACM*, 53(2):37–42, Feb. 2010.

[16] Ray Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22, 1965.

[17] John von Neumann. Zur Theorie des Gesellschaftsspiele. *Math. Ann.*, 100:295–320, 1928.

[18] Hal Wasserman and Manuel Blum. Software reliability via run-time result-checking. *Journal of the ACM*, 44(6):826–849, 1997.