Probability and Complexity: Two Sides of the Same Coin

Kamaludin Dingle

May 2022

Article prepared for *Significance* (Royal Statistical Society)

What is the relationship between Alan Turing, a monkey with a keyboard, and the symmetric arrangement of flower petals? Sounds like the beginning of a joke perhaps, but in fact the answer lies in algorithmic probability, the mathematical theory that posits an intrinsic connection between the complexity of a pattern and its probability. This theory suggests the intriguing possibility that some shape and pattern probabilities can be estimated directly by examining the shapes and patterns themselves. To see how, we take a step back in history, nearly 100 years.

After graduating in 1934 with a degree in mathematics from Cambridge University, Alan Turing put his mind to a central problem of the time, namely the mathematician David Hilbert's decidability problem. The challenge concerned whether it was possible to decide if a given mathematical proposition was provable or not. Turing wanted to show that there were some well-defined mathematical questions which could not be settled by computation. In tackling this problem, he developed an abstract and general computation device which could be used to frame and study such questions. His theoretical device was constructed to be capable of implementing any conceivable set of instructions (i.e. algorithms or programs). The instructions were on an input tape, which were processed via a series of mechanical operations, and then the output of the computation was printed onto a separate output tape. He envisioned both the input tape and the output tape to be written as binary strings, meaning sequences of 0's and 1's. Visually, one can imagine feeding a long strip of paper containing a sequence of 0's and 1's into a box, while emerging from the other side of the box streams another piece of paper with a new sequence of 0's and 1's. This device came to be called a Turing machine, or universal Turing machine, highlighting its capacity to implement all manner of calculations and algorithms. Although not intended as a practical device, Turing's work in combination with that of others like Kurt Gödel, Alonzo Church, and Emil Post, founded theoretical computer science and thereby lead eventually to the computers which we use today.

In the 1960's, Ray Solomonoff, Gregory Chaitin, and Andrei Kolmogorov independently proposed an absolute measure of complexity, now known as Kolmogorov complexity, $K(x)$, which aims to quantify the information content of a pattern or sequence x. Loosely, $K(x)$ is the length of the shortest program that runs on a universal Turing machine and produces x, or put differently, the size of the compressed version of the sequence x. (Readers will be familiar with compressing a file when attaching it to an email, for example.) This compression-based approach is premised on the idea that if x can be generated or described by a short program, it must be simple or regular; on the other hand, if describing x requires a long program, x must have a complicated, irregular pattern. Let's look at two example sequences: Taking

$$x = \text{abababababababababababababab} \tag{1}$$

then x is simple because it is just a repeating pattern of "ab" 15 times, and hence x is very compressible and would have a small value for $K(x)$. Now instead taking

$$y = \text{baabbbaaababababbbabbabaabaabbaa} \tag{2}$$

this random sequence y has no patterns by which we could compress it, so it is complex with a high value for $K(y)$.

Because Kolmogorov complexity concerns measuring randomness and patterns in sequences the theory has overlap with, and implications for, statistics and probability. For example, Kolmogorov complexity is used to help form a theoretical basis for what we mean by "pattern", which is important, given that statistics is largely about finding patterns. One remarkable result from studying universal Turing machines and Kolmogorov complexity is that it is not possible — even in principle — to prove that a sequence is random. In other words, if it is non-random, we can often demonstrate this by highlighting patterns in the sequence; but it is not possible to prove that it is random due to reduction to logical problems of self-reference (Box 1). Given our modern-day interest in statistics, data science, and machine learning, having an underlying theory of pattern is valuable and may help in developing new and better ways to detect or measure patterns in data.

Box 1: Is it random? Can you be sure?
Given a sequence x of data values, a natural question for a statistician to ask it whether the sequence is purely random, or if instead it contains (perhaps hidden) patterns. Imagine we had tried a battery of randomness tests on x, and it passed all those tests. Does that prove the sequence is random? Unfortunately, it does not because we may have chosen the wrong tests. What might help is if some computational method could try out *all* possible tests on the series; but can this be done? Per Martin-Löf proved that if any statistical test for randomness declared a sequence non-random, then the sequence has low Kolmogorov complexity. Loosely, a sequence x of n bits is random if $K(x) \approx n$ bits and non-random if it can be compressed to a shorter program of $K(x) < n$ bits. Essentially, "non-random" is the same as being compressible. An apparently promising line of attack to proving a sequence is random would be to calculate $K(x)$ and compare this value to n. However, this is impossible in general, because $K(x)$ is uncomputable, meaning that no algorithm can calculate the value precisely. This uncomputability is related to the famous *halting problem*, which states that it is not possible in general to decide in advance whether or not a program will halt and produce some output, or just keep running forever. So, we can never be sure that we have tried enough tests or waited long enough for a test to finish running. Hence a sequence can never be proved a random. The proposal to investigate all possible patterns in a sequence leads into the murky waters of logical problems due to self-referential statements.

Building on earlier work by Solomonoff, in 1974 Leonid Levin explored the question of what happens when a universal Turing machine runs a random binary program made up of 0's and 1's, for example generated by coin flips. Of course, many random programs will contain nonsensical instructions or other errors, or possibly run forever in an infinite loop, and hence not print any output. But occasionally some of these random programs will run and generate some output. Writing $P(x)$ for the probability of generating sequence pattern x from running a random program, Levin showed that a deep and direct connection exists between probability and complexity as follows

$$P(x) \approx 2^{-K(x)} \tag{3}$$

This equation says that the probability $P(x)$ of generating a pattern x can be calculated by using its Kolmogorov complexity $K(x)$, and implies that simple patterns will have high probability, whereas complex patterns will have (exponentially) lower probability. Probability estimates based on random programs are called *algorithmic probability* (Box 2), and the probability distribution is known as the *universal distribution*.

Box 2: Algorithmic probability, in more detail.
As part a research program on the mathematical foundations of induction, in the 1960's Solomonoff had already introduced the idea of algorithmic probability and showed that $P(x) \geq 2^{-K(x)}$. To derive this result, observe that we know there is a program p of length $K(x)$ bits that yields the output x when run on a universal Turing machine. Now, if you have a random binary program made up of 0's and 1's, the probability that the first random bit equals to the first bit of p is $1/2$, similarly the probability of getting the second bit of p is $1/2$, etc. To get all $K(x)$ bits right, you need to multiply all the probabilities $1/2 \times 1/2 \times \cdots \times 1/2$ a total of $K(x)$ times, which gives $(1/2)^{K(x)} = 2^{-K(x)}$. Proving the full result of $P(x) \approx 2^{-K(x)}$ takes more work, because x can be generated by many different programs, not just by the program p. So we might have thought that $P(x) \gg 2^{-K(x)}$ for some x due to adding up the probabilities from all these other programs. However, Levin showed this is not the case. Indeed, if some complex patterns were to have high probability, it would be as if information was being created by the computation process itself, which cannot happen.

This algorithmic probability equation is surprising because it connects two apparently unrelated concepts — probability and data compression — in a direct and simple way. It is also striking because it shows that you may not need to know the details of how x was generated in order to estimate its probability: Just by looking at the pattern itself and estimating its complexity via compression, you can guess its probability.

An intuition for this connection can be grasped with a little help from a monkey with a keyboard. Let's suppose Fred, a monkey, has got his hands on a computer keyboard and wants to have a go at writing some computer program in the coding language Python (other common languages like C++ or Fortran etc. would also do fine). Not having brushed up on his computer programming, Fred decides to randomly hit keys and hope for the best. What kinds of outputs are most likely? Using the example sequences introduced above, we can see that to generate the simple sequence x = abababababababababababababab Fred would just need to be lucky enough to write

```
print(15*'ab')
```
(4)

which is a short program, that is possible to construct due to the simple pattern in x. In contrast, to generate the complex irregular pattern
y =baabbbaaababababbbabbabaabaabbaa requires a long program of

```
print('baabbbaaababababbbabbabaabaabbaa')
```
(5)

because there are no regularities or repetitions to exploit which would allow for a shorter program. The key idea is that because the short program requires much fewer correct characters — much less information — it is more easily produced by chance than the long program. So a given simple, regular pattern is much more likely than a given complex, irregular pattern. We call this *simplicity bias*.

The prediction that simple and regular patterns are more likely is in stark contrast to what you would see if instead a sequence of a's and b's was made by mere random coin flips, because coin flips are much more likely to produce complex "random looking" sequences than simple regular sequences. So, we see that the outputs of random computer programs, governed by algorithmic probability, have intriguing and perhaps unintuitive properties.

Now that we have the basic idea of the complexity-probability connection, let's survey some applications (which are, simply put, biased towards my personal interests). It turns out that directly applying algorithmic probability results in the real world is not straightforward for a number of reasons, including the fact that $K(x)$ cannot be precisely calculated (Box 1), strictly speaking the results are asymptotic (i.e. they only apply for large complexity values), and assuming the presence of universal Turing machines is problematic.

Undaunted by these challenges, several researchers have tried to approximate or otherwise apply these results in different fields. One such application was in 2018 when Ard Louis, Chico Camargo and I wrote a paper in which we presented a practically applicable version of Levin's equation:

$$P(x) \leq 2^{-a\tilde{K}(x)-b} \tag{6}$$

This equation applies to real-world mathematical functions (assuming some conditions), not just computer program outputs. In this equation, a and b are constants that are often easy to estimate, and $\tilde{K}(x)$ is an approximation to the Kolmogorov complexity based on standard compression algorithms. We used this equation to study simplicity bias in a range of examples, including computer generated RNA molecule shapes, solutions of systems of (differential) equations, series from financial models, a matrix multiplication map, and computer-generated plant shapes. What we found in practice is that just using an estimate of complexity of a pattern or shape, we could accurately predict the upper bound on $P(x)$, meaning the highest possible probability that the pattern x could have. Further, as expected, higher probability patterns and shapes were simple, and complex patterns and shapes had much lower probabilities.

In Fig 1 (taken from Dingle, Camargo, Louis (2018) *Nature Communications*) we show an example of a complexity-probability plot for computer-generated plant shapes. The black line is the upper bound prediction just based on the complexities of the shapes.

The highest probability value for each complexity closely follows the black line, and so we see that the upper bound prediction is quite accurate, and non-trivial probability predictions can be made just from estimating the complexity of the shapes. There are lots of shapes which have low complexity and low probability, in contrast to what we would have seen based on Levin's equation above. Nonetheless we showed that randomly generated output patterns are very likely to be close to the bound.

More recently, my collaborators and I sought to apply algorithmic probability to natural biological shapes. Viewing DNA sequences as 'programs' which are 'computed' to generate biological forms (e.g. molecular shapes, snail shell

spirals, flower petal arrangements, etc.) then this suggests biology as an ideal setting for the mathematics of algorithms to apply. Based on the short-programs argument and our practical version of algorithmic probability described above, we argued that nature has an inbuilt bias to simple, symmetric, and regular patterns. To test our ideas, we examined bioinformatic databases of molecular shapes and also a large mathematical model of a cell cycle. We found clear evidence of simplicity bias in these real biological shapes. There may be functional advantages to simple patterns (such as flower petal symmetries) which might also explain their presence in organisms, but even without these functional aspects, nature has an intrinsic bias to simple, perhaps even beautiful, patterns.

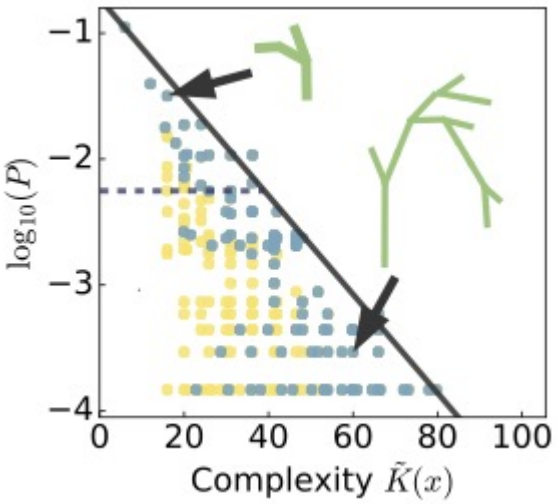

Figure 1: Simplicity bias in a model of plant shapes (L-systems). Higher probability shapes are simpler (lower complexity values), and more complex shapes have lower probabilities. Two example plant shapes are illustrated, one simple and one complex. The data points are coloured, separating the outputs which account for the top 50% of the probability from those that account for the bottom 50% of the probability for each complexity value

An additional interesting result from our biological study, beyond explaining simplicity bias, is that despite the fact the shapes and patterns in organisms result from many different factors such as functional relevance and the role of natural selection, nonetheless we could still predict the frequency (or probability) of the different shapes just based on their estimated complexity values. It is quite striking that this works even in the 'messy' world of biology. It is an open question as to whether other aspects of biology and other sciences could be explained or predicted by this fundamental complexity-probability relation.

Moving to a different application area, there is currently a lot of interest in machine learning and artificial intelligence because many believe that applications of these fields to every aspect of life — from health, to business, to warfare — will literally change the world. Despite the obvious successes of these forms of statistical learning from data, an open mathematical problem is to explain why they work so well in practice. Towards addressing this problem, Guillermo Valle Pérez, Chico Camargo, and Ard Louis used our practical version of algorithmic probability described above to argue that the unexpected success of machine learning algorithms (specifically neural networks) to learn and extrapolate patterns in data is partly explained by simplicity bias. The idea is that because natural data display simplicity bias, and the machine learning algorithms are also biased towards simpler functions, this makes the job of learning patterns easier. In this sense, the proposal is that there is an in-built Occam's razor in artificial intelligence methods.

Their work relates to that of Marcus Hutter, who has shown that algorithmic probability plays a fundamental role in his theory of universal artificial intelligence, describing how an intelligent agent should learn and interact with its (possibly unknown) environment to maximise its relevant reward. What acting "intelligently" means can be vague, but Hutter specifies it as making good inferences (which algorithmic probability helps you to do) combined with deciding what actions to take based on your prediction of what will happen next. Because of the fundamental importance of data compression in this general theory of intelligence, to inspire more work in this area Hutter has offered a large cash reward for an ongoing public competition to compress the data in Wikipedia ever more efficiently.

Finally, looking to future applications, this year we have organized a conference — coincidentally based in the Alan Turing Institute, London — with the aim of developing new results and theory in data science and machine learning based on Kolmogorov complexity, algorithmic probability, and other aspects of algorithmic information theory.

To conclude this article, we return to the opening question about the relationship between Turing, a monkey with a keyboard, and the symmetries of flowers. Hopefully the reader will now see that these are indeed connected, namely that a monkey randomly programming a Turing machine is more likely to produce a simple and symmetric pattern like those of a flower, rather than some complex and irregular form. We saw how predicting the probability of a shape or pattern directly from its complexity value not only has a sound theoretical basis, but can also work in practice. A central 'take home' message is that while a sequence of random coin flips would typically show complex patterns and rarely simple ones, for patterns computed from random programs the opposite occurs: simple patterns are typical, and complex patterns are rarer. Finally, algorithmic probability can, continues to, and ought to, be applied in myriad ways in the fields of probability and statistics.

About the author:
Kamaludin Dingle obtained his PhD in mathematical biology from Oxford University. He is an Associate Professor of Applied Mathematics and Quantitative Biology at the Gulf University for Science and Technology, Kuwait. From summer 2022 he is on research leave, working on theoretical biology at Cambridge University, and time series forecasting at the California Institute of Technology (Caltech).

Some useful references:
The miraculous universal distribution
The Mathematical Intelligencer 19, 7–15 (1997)
Walter Kirchherr, Ming Li, and Paul Vitányi

Universal artificial intelligence: Sequential decisions based on algorithmic probability
Marcus Hutter. Springer
Science & Business Media, 2004.

Algorithmic probability
Scholarpedia, 2(8):2572
Marcus Hutter et al. (2007)

Input-output maps are strongly biased towards simple outputs
Nature Communications (2018) 9(761)
Kamaludin Dingle, Chico Camargo, Ard Louis

Deep learning generalizes because the parameter-function map is biased towards simple functions
Guillermo Valle Pérez, Chico Q. Camargo, Ard Louis (2018)
https://arxiv.org/abs/1805.08522

Coding-theorem like behaviour and emergence of the universal distribution from resource-bounded algorithmic probability
International Journal of Parallel, Emergent and Distributed Systems, 34(2):161–180, 2019
Hector Zenil, Liliana Badillo, Santiago Hernández-Orozco, and Francisco Hernández-Quiroz

Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution
Proceedings of the National Academy of Sciences, 2022; 119 (11)
Iain Johnston, Kamaludin Dingle, Sam Greenbury, Chico Camargo, Jonathan Doye, Sebastian Ahnert, and Ard Louis.