Algorithmic complexity for psychology: a user-friendly implementation of the coding theorem method

**Nicolas Gauvrit · Henrik Singmann ·
Fernando Soler-Toscano · Hector Zenil**

Abstract Kolmogorov-Chaitin complexity has long been believed to be impossible to approximate when it comes to short sequences (e.g. of length 5-50). However, with the newly developed *coding theorem method* the complexity of strings of length 2-11 can now be numerically estimated. We present the theoretical basis of algorithmic complexity for short strings (ACSS) and describe an R-package providing functions based on ACSS that will cover psychologists' needs and improve upon previous methods in three ways: (1) ACSS is now available not only for binary strings, but for strings based on up to 9 different symbols, (2) ACSS no longer requires time-consuming computing, and (3) a new approach based on ACSS gives access to an estimation of the complexity of strings of any length. Finally, three illustrative examples show how these tools can be applied to psychology.

N. Gauvrit (✉)
CHArt (PARIS-reasoning), Ecole Pratique des Hautes Etudes, Paris, France
e-mail: ngauvrit@me.com

H. Singmann
Institut für Psychologie, Albert-Ludwigs-Universität Freiburg, Freiburg, Germany

F. Soler-Toscano
Grupo de Lógica, Lenguaje e Información, Universidad de Sevilla, Sevilla, Spain

H. Zenil
Unit of Computational Medicine, Center for Molecular Medicine, Karolinska Institute, Stockholm, Sweden

Randomness and complexity are two concepts which are intimately related and are both central to numerous recent developments in various fields, including finance (Taufemback et al., 2011, Brandouy et al., 2012), linguistics (Gruber, 2010; Naranan, 2011), neuropsychology (Machado et al., 2010; Fernández et al., 2011, 2012), psychiatry (Yang and Tsai, 2012; Takahashi 2013), genetics (Yagil, 2009; Ryabko et al. 2013), sociology (Elzinga, 2010) and the behavioral sciences (Watanabe et al., 2003; Scafetta et al., 2009). In psychology, randomness and complexity have recently attracted interest, following the realization that they could shed light on a diversity of previously undeciphered behaviors and mental processes. It has been found, for instance, that the subjective difficulty of a concept is directly related to its "boolean complexity", defined as the shortest logical description of a concept (Feldman, 2000, 2003, 2006). In the same vein, visual detection of shapes has been shown to be related to contour complexity (Wilder et al., 2011).

More generally, perceptual organization itself has been described as based on simplicity or, equivalently, likelihood (Chater, 1996; Chater and Vitányi, 2003), in a model reconciling the complexity approach (perception is organized to minimize complexity) and a probability approach (perception is organized to maximize likelihood), very much in line with our view in this paper. Even the perception of similarity may be viewed through the lens of (conditional) complexity (Hahn et al., 2003).

Randomness and complexity also play an important role in modern approaches to selecting the "best" among a set of candidate models (i.e., model selection; e.g., Myung et

al., 2006; Kellen et al., 2013), as discussed in more detail below in the section called "Relationship to complexity based model selection".

Complexity can also shed light on short term memory storage and recall, more specifically, on the process underlying *chunking*. It is well known that the short term memory span lies between 4 and 7 items/chunks (Miller, 1956; Cowan, 2001). When instructed to memorize longer sequences of, for example, letters or numbers, individuals employ a strategy of subdividing the sequence into chunks (Baddeley et al., 1975). However, the way chunks are created remains largely unexplained. A plausible explanation might be that chunks are built via minimizing the complexity of each chunk. For instance, one could split the sequence "AAABABABA" into the two substrings "AAA" and "BABABA". Very much in line with this idea, Mathy and Feldman (2012) provided evidence for the hypothesis that chunks are units of "maximally compressed code". In the above situation, both "AAA" and "BABABA" are supposedly of low complexity, and the chunks tailored to minimize the length of the resulting compressed information.

Outside the psychology of short term memory, the complexity of pseudo-random human-generated sequences is related to the strength of executive functions, specifically to inhibition or sustained attention (Towse and Cheshire, 2007). In random generation tasks, participants are asked to generate random-looking sequences, usually involving numbers from 1 to 6 (dice task) or digits between 1 and 9. These tasks are easy to administer and very informative in terms of higher level cognitive abilities. They have been used in investigations in various areas of psychology, such as visual perception (Cardaci et al., 2009), aesthetics (Boon et al., 2011), development (Scibinetti et al., 2011; Pureza et al., 2013), sport (Audiffren et al., 2009), creativity (Zabelina et al., 2012), sleep (Heuer et al., 2005; Bianchi and Mendez, 2013), and obesity (Crova et al., 2013), to mention a few. In neuropsychology, random number generation tasks and other measures of behavioral or brain activity complexity have been used to investigate different disorders, such as schizophrenia (Koike et al., 2011), autism (Lai et al., 2010; Maes et al., 2012; Fournier et al., 2013), depression (Fernandez et al., 2009), PTSD (Pearson and Sawyer, 2011; Curci et al., 2013), ADHD (Sokunbi et al., 2013), OCD (Bédard et al., 2009), hemispheric neglect (Loetscher and Brugger, 2009), aphasia (Proios et al., 2008), and neurodegenerative syndromes such as Parkinson's and Alzheimer's disease (Brown and Marsden, 1990; Hahn et al., 2012).

Perceived complexity and randomness are also of the utmost importance within the "new paradigm psychology of reasoning" (Over, 2009). As an example, let us consider the representativeness heuristic (Tversky and Kahneman, 1974). Participants usually believe that the sequence "HTHTHTHT" is less likely to occur than the sequence "HTHHTHTT" when a fair coin is tossed 8 times. This of course is mathematically wrong, since all sequences of 8 heads or tails, including these two, share the same probability of occurrence ($1/2^8$). The "old paradigm" was concerned with finding such biases and attributing irrationality to individuals (Kahneman et al., 1982). In the "new paradigm" on the other hand, researchers try to discover the ways in which sound probabilistic or Bayesian reasoning can lead to the observed errors (Manktelow and Over, 1993; Hahn and Warren, 2009).

We can find some kind of rationality behind the wrong answer, by assuming that individuals do not estimate the probability that a fair coin will produce a particular string s but rather the "inverse" probability that the process underlying this string is mere chance. More formally, if we use s to denote a given string, R to denote the event where a string has been produced by a random process, and D to denote the complementary event where a string has been produced by a non-random (or deterministic) process, then individuals may assess $P(R|s)$ instead of $P(s|R)$. If they do so within the framework of formal probability theory (and the new paradigm postulates that individuals tend to do so), then their estimation of the probability should be such that Bayes' theorem holds:

$$P(R|s) = \frac{P(s|R)P(R)}{P(s|R)P(R) + P(s|D)P(D)}. \tag{1}$$

Alternatively, we could assume that individuals do not estimate the complete inverse $P(R|s)$ but just the posterior odds of a given string being produced by a random rather than a deterministic process (Williams and Griffiths, 2013). Again, these odds are given by Bayes' theorem:

$$\frac{P(R|s)}{P(D|s)} = \frac{P(s|R)}{P(s|D)} \times \frac{P(R)}{P(D)}. \tag{2}$$

The important part of Eq. 2 is the first term on the right-hand side, as it is a function of the observed string s and independent of the prior odds $P(R)/P(D)$. This likelihood ratio, also known as the Bayes factor (Kass and Raftery, 1995), quantifies the evidence brought by the string s based on which the prior odds are changed. In other words, this part corresponds to the "amount of evidence [s] provides in favor of a random generating process" (Hsu et al., 2010).

The numerator of the Bayes factor, $P(s|R)$, is easily computed, and amounts to $1/2^8$ in the example given above. However the other likelihood, the probability of s given that it was produced by an (unknown) deterministic process, $P(s|D)$, is more problematic. Although this probability has been informally linked to complexity, to the best of our knowledge no formal account of that link has ever been provided in the psychological literature, although some authors have suggested such a link (e.g., Chater, 1996). As we will

see, however, computer scientists have fruitfully addressed this question (Solomonoff, 1964a, b; Levin, 1974). One can think of $P(s|D)$ as the probability that a randomly selected (deterministic) algorithm produces s. In this sense, $P(s|D)$ is none other than the so-called *algorithmic probability of s*. This probability is formally linked to the algorithmic complexity $K(s)$ of the string s by the following formula (see below for more details):

$$K(s) \approx -\log_2(P(s|D)). \tag{3}$$

A normative measure of complexity and a way to make sense of $P(s|D)$ are crucial to several areas of research in psychology. In our example concerning the representativeness heuristic, one could see some sort of rationality in the usually observed behavior if in fact the complexity of $s_1 = $ "HTHTHTHT" were lower than that of $s_2 = $ "HTHHTHTT" (which it is, as shown below). Then following Eq. 3, $P(s_1|D) > P(s_2|D)$. Consequently, the Bayes factor for a string being produced by a random process would be larger for s_2 than for s_1. In other words, even when ignoring the question of the priors for a random versus deterministic process (which are inherently subjective and debatable) s_2 provides more evidence for a random process than s_1.

Researchers have hitherto relied on an intuitive perception of complexity, or in the last decades developed and used several tailored measures of randomness or complexity (Towse, 1998; Barbasz et al., 2008; Schulter et al., 2010; Williams and Griffiths, 2013, Hahn and Warren, 2009) in the hope of approaching algorithmic complexity. Because all these measures rely upon choices that are partially subjective and each focuses on a single characteristic of chance, they have come under strong criticism (Gauvrit et al., 2013). Among these measures, some have a sound mathematical basis, but focus on particular features of randomness. For that reason, contradictory results have been reported (Wagenaar, 1970; Wiegersma, 1984). The mathematical definition of complexity, known as Kolmogorov-Chaitin complexity theory (Kolmogorov, 1965; Chaitin, 1966), or simply algorithmic complexity, has been recognized as the best possible option by mathematicians Li and Vitányi (2008) and psychologists Griffiths and Tenenbaum (2003, 2004). However, because algorithmic complexity was thought to be impossible to approximate for the short sequences we usually deal with in psychology (sequences of 5–50 symbols, for instance), it has seldom been used.

In this article, we will first briefly describe algorithmic complexity theory and its deep links with algorithmic probability (leading to a formal definition of the probability that an unknown deterministic process results in a particular observation s). We will then describe the practical limits of algorithmic complexity and present a means to overcome them, namely the coding theorem method, the root of *algorithmic complexity for short strings* (ACSS). A new set of tools, bundled in a package for the statistical programming language R (R Core Team, 2014) and based on ACSS, will then be described. Finally, three short applications will be presented for illustrative purposes.

Algorithmic complexity for short strings

Algorithmic complexity

As defined by Alan Turing, a universal Turing machine is an abstraction of a general-purpose computing device capable of running any computer program. Among universal Turing machines, some are *prefix free*, meaning that they only accept programs finishing with an "END" symbol. The algorithmic complexity (Kolmogorov, 1965; Chaitin, 1966) – also called Kolmogorov or Kolmogorov-Chaitin complexity – of a string s is given by the length of the shortest computer program running on a universal prefix-free Turing machine U that produces s and then halts, or formally written, $K_U(s) = \min\{|p| : U(p) = s\}$. From the *invariance theorem* (Calude, 2002; Li and Vitányi, 2008), we know that the impact of the choice of U (that is, of a specific Turing machine), is limited and independent of s. It means that for any other universal Turing machine U', the absolute value of $K_U(s) - K_{U'}(s)$ is bounded by some constant $C_{U,U'}$ which depends on U and U' but not on the specific s. So $K(s)$ is usually written instead of $K_U(s)$.

More precisely, the invariance theorem states that $K(s)$ computed on two different Turing machine will differ at most by an additive constant c, which is independent of s, but that can be arbitrary large. One consequence of this theorem is that there actually are infinitely many different complexities, depending on the Turing machine. Talking about "the algorithmic complexity" of a string is a shortcut. This theorem also guarantees that asymptotically (or for long strings), the choice of the Turing machine will have limited impact. However, for short strings we are considering here, the impact could be important, and different Turing machines can yield different values, or even different orders. This limitation is not due to technical reasons, but the fact that there is no objective default universal Turing machine one could pick to compute "the" complexity of a string. As we will explain below, our approach seeks to overcome this difficulty by defining what could be thought of as a "mean complexity" as computed with random Turing machines. Because we do not choose a particular Turing machine but sample the space of all possible Turing machine (running on a blank tape), the result will be an objective estimation of algorithmic complexity, although it will of course differ from most specific K_U.

Algorithmic complexity gave birth to a definition of randomness. To put it in a nutshell, a string is random if it is complex (i.e. exhibit no structure). Among the most striking results of algorithmic complexity theory is the convergence in definitions of randomness. For example, using martingales, Schnorr (1973) proved that Kolmogorov random (complex) sequences are effectively unpredictable and vice versa; Chaitin (2004) proved that Kolmogorov random sequences pass all effective statistical tests for randomness and vice versa, and are therefore equivalent to Martin-Löf randomness (Martin-Löf, 1966), hence the general acceptance of this measure. K has become the accepted ultimate universal definition of complexity and randomness in mathematics and computer science(Downey and Hirschfeldt, 2008; Nies, 2009; Zenil, 2011a).

One generally offered caveat regarding K is that it is *uncomputable*, meaning there is no Turing machine or algorithm that, given a string s, can output $K(s)$, the length of the shortest computer program p that produces s. In fact, the theory shows that no computable measure can be a universal complexity measure. However, it is often overlooked that K is *upper semi-computable*. This means that it can be effectively approximated from above. That is, that there are effective algorithms, such as lossless compression algorithms, that can find programs (the decompressor + the data to reproduce the string s in full) giving upper bounds of Kolmogorov complexity. However, these methods are inapplicable to short strings (of length below 100), which is why they are seldom used in psychology. One reason why compression algorithms are unadapted to short strings is that compressed files include not only the instructions to decompress the string, but also file headers and other data structures. It makes that for a short string s, the size of a compressed text file with just s is longer than $|s|$. Another reason is that, as it is the case with the complexity K_U associated with a particular Turing machine U, the choice of the compression algorithm is crucial for short strings, because of the invariance theorem.

Algorithmic probability and its relationship to algorithmic complexity

A universal prefix-free Turing machine U, can also be used to define a probability measure m on the set of all possible strings by setting $m(s) = \sum_{p:U(p)=s} 1/2^{|p|}$, where p is a program of length $|p|$, and $U(p)$ the string produced by the Turing machine U fed with program p. The Kraft inequality (Calude, 2002) guarantees that $0 \leq \sum_s m(s) < 1$. The number $m(s)$ is the probability that a randomly selected deterministic program will produce s and then halt, or simply the *algorithmic probability of s,* and provides a formal definition of $P(s|D)$, where D stands for a

generic deterministic algorithm (for a more detailed description see Gauvrit et al., 2013). Numerical approximations to $m(s)$ using standard Turing machines have shed light on the stability and robustness of $m(s)$ in the face of changes in U, providing examples of applications to various areas leading to semantic measures (Cilibrasi and Vitányi, 2005, 2007), which today are accepted as regular methods in areas of computer science and linguistics, to mention but two disciplines.

Recent work (Delahaye and Zenil, 2012; Zenil, 2011b) has suggested that approximating $m(s)$ could in practice be used to approximate K for short strings. Indeed, the *algorithmic coding theorem* (Levin, 1974) establishes the connection as $K(s) = -\log_2 m(s) + O(1)$, where $O(1)$ is bounded independently of s. This relationship shows that strings with low $K(s)$ have the highest probability $m(s)$, while strings with large $K(s)$ have a correspondingly low probability $m(s)$ of being generated by a randomly selected deterministic program. This approach, based upon and motivated by algorithmic probability, allows us to approximate K by means other than lossless compression, and has been recently applied to financial time series (Brandouy et al., 2012; Zenil and Delahaye, 2011) and in psychology (e.g. Gauvrit et al., 2014). The approach is equivalent to finding the best possible compression algorithm with a particular computer program enumeration.

Here, we extend a previous method addressing the question of binary strings' complexity (Gauvrit et al., 2013) in several ways. First, we provide a method to estimate the complexity of strings based on any number of different symbols, up to 9 symbols. Second, we provide a fast and user-friendly algorithm to compute this estimation. Third, we also provide a method to approximate the complexity (or the local complexity) of strings of medium length (see below).

The invariance theorem and the problem with short strings

The invariance theorem does not provide a reason to expect $-\log_2(m(s))$ and $K(s)$ to induce the same ordering over short strings. Here, we have chosen a simple and standard Turing machine model (the Busy Beaver model; Rado, 1962)[1] in order to build an output distribution based on the seminal concept of algorithmic probability. This output distribution then serves as an objective complexity measure producing results in agreement both with intuition and with $K(s)$ to which it will converge for long strings guaranteed by the invariance theorem. Furthermore, we have found that estimates of $K(s)$ are strongly correlated to those produced

---

[1] A demonstration is available online at `http://demonstrations.wolfram.com/BusyBeaver/`

by lossless compression algorithms as they have traditionally been used as estimators of $K(s)$ (compressed data is a sufficient test of non-randomness, hence of low $K(s)$), when both techniques (the coding theorem method and lossless compression) overlap in their range of application for medium size strings in the order of hundreds to 1K bits.

The lack of guarantee in obtaining $K(s)$ from $-\log_2(m(s))$ is a problem also found in the most traditional method to estimate $K(s)$. Indeed, there is no guarantee that some lossless compressor algorithm will be able to compress a string that is compressible by some (or many) other(s). We do have statistical evidence that, at least with the Busy Beaver model, extending or reducing the Turing machine sample space does not impact the order (Zenil et al., 2012; Soler-Toscano et al., 2013, 2014). We also found a correlation in output distribution using very different computational formalisms, e.g. cellular automata and Post tag systems (Zenil and Delahaye, 2010). We have also shown (Zenil et al., 2012) that $-\log_2(m(s))$ produces results compatible with compression methods $K(s)$, and strongly correlates to direct $K(s)$ calculation (length of first shortest Turing machine found producing s; Soler-Toscano et al., 2013).

The coding theorem method in practice

The basic idea at the root of the coding theorem method is to compute an approximation of $m(s)$. Instead of choosing a particular Turing machine, we ran a huge sample of Turing machines, and saved the resulting strings. The distribution of resulting strings gives the probability that a randomly selected Turing machine (equivalent to a universal Turing machine with a randomly selected program) will result in a given string. It therefore approximates $m(s)$. From this, we estimate an approximation K of the algorithmic complexity of any string s using the equation $K(s) = -\log_2(m(s))$. To put it in a nutshell again: A string is complex and henceforth random if the likelihood of it being produced by a randomly selected algorithm is low, which we estimated as described in the following.

To actually build the frequency distributions of strings with different numbers of symbols, we used a Turing machine simulator, written in C++, running on a supercomputer of middle-size at CICA (Centro Informático Científico de Andalucía). The simulator runs Turing machines in (n, m) (n is the number of states of the Turing machine, and m the number of symbols it uses) over a blank tape and stores the output of halting computations. For the generation of random machines, we used the implementation of the Mersenne Twister in the Boost C++ library.

Table 1 summarizes the size of the computations to build the distributions. The data corresponding to (5, 2) comes from a full exploration of the space of Turing machines with

Table 1 Data of the computations to build the frequency distributions

| (n, m)	Steps	Machines	Time
(5,2)	500	9 658 153 742 336	450 days
(4,4)	2000	3.34×10^{11}	62 days
(4,5)	2000	2.14×10^{11}	44 days
(4,6)	2000	1.8×10^{11}	41 days
(4,9)	4000	2×10^{11}	75 days

5 states and 2 symbols, as explained in Soler-Toscano et al. (2013, 2014). All other data, previously unpublished, correspond to samples of machines. The second column is the runtime cut. As the detection of non-halting machines is an undecidable problem, we stopped the computations exceeding that runtime. To determine the runtime bound, we first picked a sample of machines with an apt runtime T. For example, in the case of (4, 4), we ran 1.68×10^{10} machines with a runtime cut of 8000 steps. For halting machines in that sample, we built the runtime distribution (Fig. 1). Then we chose a runtime lower than T with an accumulated halting probability very close to 1. That way we chose 2000 steps for (4, 4). In Soler-Toscano et al. (2014) we argued that, following this methodology, we were able to cover the vast majority of halting machines.

The third column in Table 1 is the size of the sample, that is, the machines that were actually run at the C++ simulator. After these computations, several symmetric completions were applied to the data, so in fact the number of machines represented in the samples is greater. For example, we only considered machines moving to the right at the initial transition. So we complemented the set of output strings with their reversals. More details about the shortcuts to reduce the computations and the completions can be found elsewhere (Soler-Toscano et al., 2013). The last column in Table 1 is an estimate of the time the computation would have taken on a single processor at the CICA supercomputer. As we used between 10 and 70 processors, the actual times the computations took were shorter. In the following paragraphs we offer some details about the datasets obtained for each set of symbols.

(5,2) This distribution is the only one previously published (Soler-Toscano et al., 2014; Gauvrit et al., 2013). It consists of 99 608 different binary strings. All strings up to length 11 are included, and only 2 strings of length 12 are missing.

(4,4) After applying the symmetric completions to the 3.34×10^{11} machines in the sample, we obtained a dataset representing the output of 325 433 427 739 halting machines

Fig. 1 Runtime distribution in (4, 4). More than 99.9999 % of the Turing machines that stopped in 8000 steps or less actually halted before 2000 steps
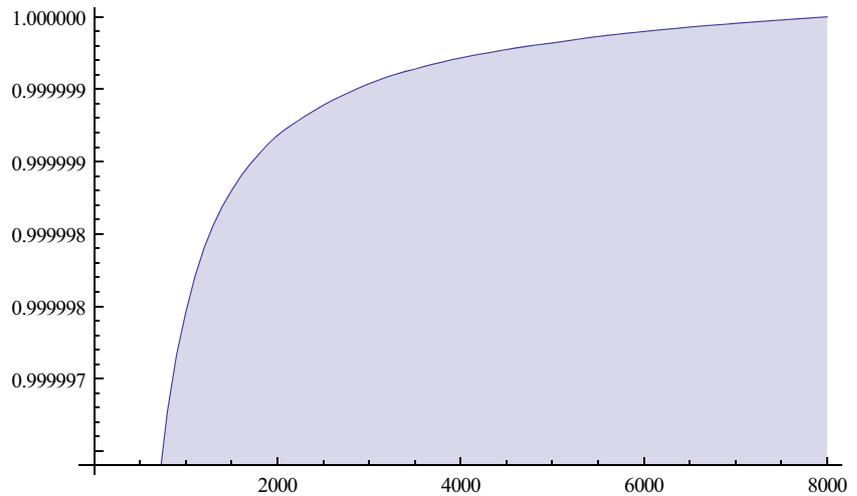
producing 17 768 208 different string patterns.[2] To reduce the file size and make it usable in practice, we selected only those patterns with 5 or more occurrences, resulting in a total of 1 759 364 string patterns. In the final dataset, all strings comprising 4 symbols up to length 11 are represented by these patterns.

(4,5) After applying the symmetric completions, we obtained a dataset corresponding to the output of 220 037 859 595 halting machines producing 39 057 551 different string patterns. Again, we selected only those patterns with 5 or more occurrences, resulting in a total of 3 234 430 string patterns. In the final dataset, all strings comprising 5 symbols up to length 10 are represented by these patterns.

(4,6) After applying the symmetric completions, we obtained a dataset corresponding to the output of 192 776 974 234 halting machines producing 66 421 783 different string patterns. Here we selected only those patterns with 10 or more occurrences, resulting in a total of 2 638 374 string patterns. In the final dataset, all strings with 6 symbols up to length 10 are represented by these patterns.

(4,9) After applying the symmetric completions, we obtained 231 250 483 485 halting Turing machines producing 165 305 964 different string patterns. We selected only those with 10 or more occurrences, resulting in a total of 5 127 061 string patterns. In the final dataset, all strings

---

[2]Two strings correspond to the same pattern if one is obtained from the other by changing the symbols, such as "11233" and "22311". The pattern is the structure of the string, in this example described as "one symbol repeated once, a second one, and a third one repeated once". Given the definition of Turing Machines, strings with the same patterns always share the same complexity.

comprising 9 symbols up to length 10 are represented by these patterns.

Summary We approximated the algorithmic complexity of short strings (ACSS) using the coding theorem method by running huge numbers of randomly selected Turing machines (or all possible Turing machines for strings of only 2 symbols) and recorded their halting state. The resulting distribution of strings obtained approximates the complexity of each string; a string that was produced by more Turing machines is less complex (or random) than one produced by fewer Turing machines. The results of these computations, one dataset per number of symbols (2, 4, 5, 6, and 9), were bundled and made freely available. Hence, although the initial computation took weeks, the distributions are now readily available for all researchers interested in a formal and mathematically sound measure of the complexity of short strings.

The `acss` packages

To make ACSS available, we have released two packages for the statistical programming language R (R Core Team, 2014) under the GPL license (Free Software Foundation, 2007) which are available at the Central R Archive Network (CRAN; http://cran.r-project.org/). An introduction to R is, however, beyond the scope of this manuscript. We recommend the use of RStudio (http://www.rstudio.com/) and refer the interested reader to more comprehensive literature (e.g., Jones et al. (2009), Maindonald and Braun (2010), and Matloff (2011)).

The first package, `acss.data`, contains only the calculated datasets described in the previous section in compressed form (the total size is 13.9 MB) and should not be used directly. The second package, `acss`, contains (a)

functions to access the datasets and obtain ACSS and (b) functions to calculate other measures of complexity, and is intended to be used by researchers in psychology who wish to analyze short or medium-length (pseudo-)random strings. When installing or loading `acss` the data-only package is automatically installed or loaded. To install both packages, simply run `install.packages("acss")` at the R prompt. After installation, the packages can be loaded with `library("acss")`.[3] The next section describes the most important functions currently included in the package.

Main functions

All functions within `acss` have some common features. Most importantly, the first argument to all functions is `string`, corresponding to the string or strings for which one wishes to obtain the complexity measure. This argument necessarily needs to be a character vector (to avoid issues stemming from automatic coercion). In accordance with R's general design, all functions are fully vectorized, hence `string` can be of length > 1 and will return an object of corresponding size. In addition, all functions return a named object, the names of the returned objects corresponding to the initial strings.

Algorithmic complexity and probability The main objective of the `acss` package is to implement a convenient version of algorithmic complexity and algorithmic probability for short strings. The function `acss()` returns the ACSS approximation of the complexity $K(s)$ of a string s of length between 2 and 12 characters, based on alphabets with either 2, 4, 5, 6, or 9 symbols, which we shall hereafter call K_2, K_4, K_5, K_6 and K_9. The result is thus an approximation of the length of the shortest program running on a Turing machine that would produce the string and then halt.

The function `acss()` also returns the observed probability $D(s)$ that a string s of length up to 12 was produced by a randomly selected deterministic Turing machine. Just like K, it may be based on alphabets of 2, 4, 5, 6 or 9 symbols, hereafter called D_2, D_4, ..., D_9. As a consequence of returning both K and D, `acss()` per default returns a matrix.[4] Note that the first time a function accessing `acss.data` is called within a R session, such as `acss()`, the complete data of all strings is loaded into the RAM which takes some

time (even on modern computers this can take more than 10 seconds).

```
> acss(c("aba", "aaa"))
          K.9      D.9
 aba 11.90539 0.0002606874
 aaa 11.66997 0.0003068947
```

Per default, `acss()` returns K_9 and D_9, which can be used for strings up to an alphabet of 9 symbols. To obtain values from a different alphabet one can use the `alphabet` argument (which is the second argument to all `acss` functions), which for `acss()` also accepts a vector of length > 1. If a string has more symbols than are available in the alphabet, these strings will be ignored:

```
> acss(c("01011100", "00030101"), alphabet
= c(2, 4))
K.2    K.4       D.2          D.4
01011100 22.00301 24.75269 2.379222e-07
3.537500e-08
00030101 NA 24.92399 NA 3.141466e-08
```

Local complexity When asked to judge the randomness of sequences of medium length (say 10-100) or asked to produce pseudo-random sequences of this length, the limit of human working memory becomes problematic, and individuals likely try to overcome it by subdividing and performing the task on subsequences, for example, by maximizing the local complexity of the string or averaging across local complexities (Hahn, 2014; Hahn and Warren, 2009). This feature can be assessed via the `local_complexity()` function, which returns the complexity of substrings of a string as computed with a sliding window of substrings of length `span`, which may range from 2 to 12. The result of the function as applied to a string $s = s_1 s_2 \ldots s_l$ of length l with a span k is a vector of $l - k + 1$ values. The i-th value is the complexity (ACSS) of the substring $s_{[i]} = s_i s_{i+1} \ldots s_{i+k-1}$.

As an illustration, let's consider the 8-character long string based on a 4-symbol alphabet, "aaabcbad". The local complexity of this string with `span = 6` will return $K_4(\text{aaabcb})$, $K_4(\text{aabcba})$, and $K_4(\text{abcbad})$, which equals (18.6, 19.4, 19.7):

```
> local_complexity("aaabcbad", alphabet=4,
span=6)
$aaabcbad  aaabcb    aabcba    abcbad
18.60230 19.41826 19.71587
```

Bayesian approach As discussed in the introduction, complexity may play a crucial role when combined with Bayes' theorem (see also (Williams and Griffiths 2013; Hsu et al. 2010)). Instead of the observed probability D we actually may be interested in the likelihood of a string s of length l given a deterministic process $P(s|D)$. As discussed before, this likelihood is trivial for a random process and amounts

---

[4]Remember (e.g., Eq. 3) that the measures D and K are linked by simple relations derived from the coding theorem method:

$$K(s) = -\log_2(D(s)) \qquad D(s) = 2^{-K(s)}.$$

to $P(s|R) = 1/m^l$, where m, as above, is the size of the alphabet. To facilitate the corresponding usage of ACSS, `likelihood_d()` returns the likelihood $P(s|D)$, given the actual length of s. This is done by taking $D(s)$ and normalizing it with the sum of all $D(s_i)$ for all s_i with the same length l as s (note that this also entails performing the symmetric completions). As expected in the beginning, the likelihood of "HTHTHTHT" is larger than that of "HTHHTHTT" under a deterministic process:

```
> likelihood_d(c("HTHTHTHT", "HTHHTHTT"),
                          alphabet = 2)
    HTHTHTHT   HTHHTHTT
0.010366951 0.003102718
```

With the likelihood at hand, we can make full use of Bayes' theorem, and `acss` contains the corresponding functions. One can either obtain the likelihood ratio (Bayes factor) for a random rather than deterministic process via function `likelihood_ratio()`. Or, if one is willing to make assumptions on the prior probability with which a random rather than deterministic process is responsible (i.e., $P(R) = 1 - P(D)$) one can obtain the posterior probability for a random process given s, $P(R|s)$, using `prob_random()`. The default for the prior is $P(R) = 0.5$.

```
> likelihood_ratio(c("HTHTHTHT","HTHHTHTT"),
                          alphabet = 2)
 HTHTHTHT HTHHTHTT
0.3767983 1.2589769
> prob_random(c("HTHTHTHT", "HTHHTHTT"),
                          alphabet = 2)
 HTHTHTHT HTHHTHTT
0.2736772 0.5573217
```

Entropy and second-order entropy Entropy (Barbasz et al., 2008; Shannon, 1948) has been used for decades as a measure of complexity. It must be emphasized that (first order) entropy does not capture the structure of a string, and only depends on the relative frequency of the symbols in the said string. For instance, the string "01010101010101" has a greater entropy than "0100101100100010" because the first string is balanced in terms of 0's and 1's. According to entropy, the first string, although it is highly regular, should be considered more complex or more random than the second one.

Second-order entropy has been put forward to overcome this inconvenience, but it only does so partially. Indeed, second-order entropy only captures the narrowly local structure of a string. For instance, the string "011001100110011001100110..." maximizes second-order entropy, because the four patterns 00, 01, 10 and 11 share the same frequency in this string. The fact that the sequence is the result of a simplistic rule is not taken into account. Notwithstanding these strong limitations, entropy has been extensively used. For that historical reason, `acss`

Table 2 Correlation matrix of complexity measures computed on all binary strings of length up to 11. "Ent" stands for entropy, and "Change" for change complexity

|	K2	K4	K5	K6	K9	Ent
K4	0.98					
K5	0.97	1.00				
K6	0.97	1.00	1.00			
K9	0.97	1.00	1.00	1.00		
Ent	0.32	0.36	0.36	0.37	0.38	
Change	0.69	0.75	0.75	0.75	0.75	0.50

includes two functions, `entropy()` and `entropy2()` (second order entropy).

Change complexity Algorithmic complexity for short strings is an objective and universal normative measure of complexity approximating Kolmogorov-Chaitin complexity. ACSS helps in detecting *any* computable departures from randomness. This is exactly what researchers seek when they want to assess the formal quality of a pseudorandom production. However, psychologists may also wish to assess complexity as it is perceived by human participants. In that case, algorithmic complexity may be too sensitive. For instance, there exists a (relatively) short program computing the decimals of π. However, faced with that series of digits, humans are unlikely to see any regularity: algorithmic complexity identifies as non-random some series that will look random to most humans.

When it comes to assessing *perceived* complexity, the tool developed by Aksentijevic and Gibson (2012), named "change complexity", is an interesting alternative. It is based on the idea that humans' perception of complexity depends largely on the changes between one symbol and the next. Unfortunately, change complexity is, to date, only available for binary strings. As change complexity is to our knowledge not yet included in an R-package, we have implemented it in `acss` in function `change_complexity()`.

A comparison of complexity measures

There are several complexity measures based on the coding theorem method, because the computation depends on the set of possible symbols the Turing machines can manipulate. To date, the package provides ACSS for 2, 4, 5, 6 and 9 symbols, giving 5 different measures of complexity. As we will see, however, these measures are strongly correlated. As a consequence one may use K_9 to assess the complexity of strings with 7 or 8 different symbols, or K_4 to assess the complexity of a string with 3 symbols. Thus in the end any alphabet size between 2 and 9 is available. Also, these measures are mildly correlated to change complexity, and poorly to entropy.
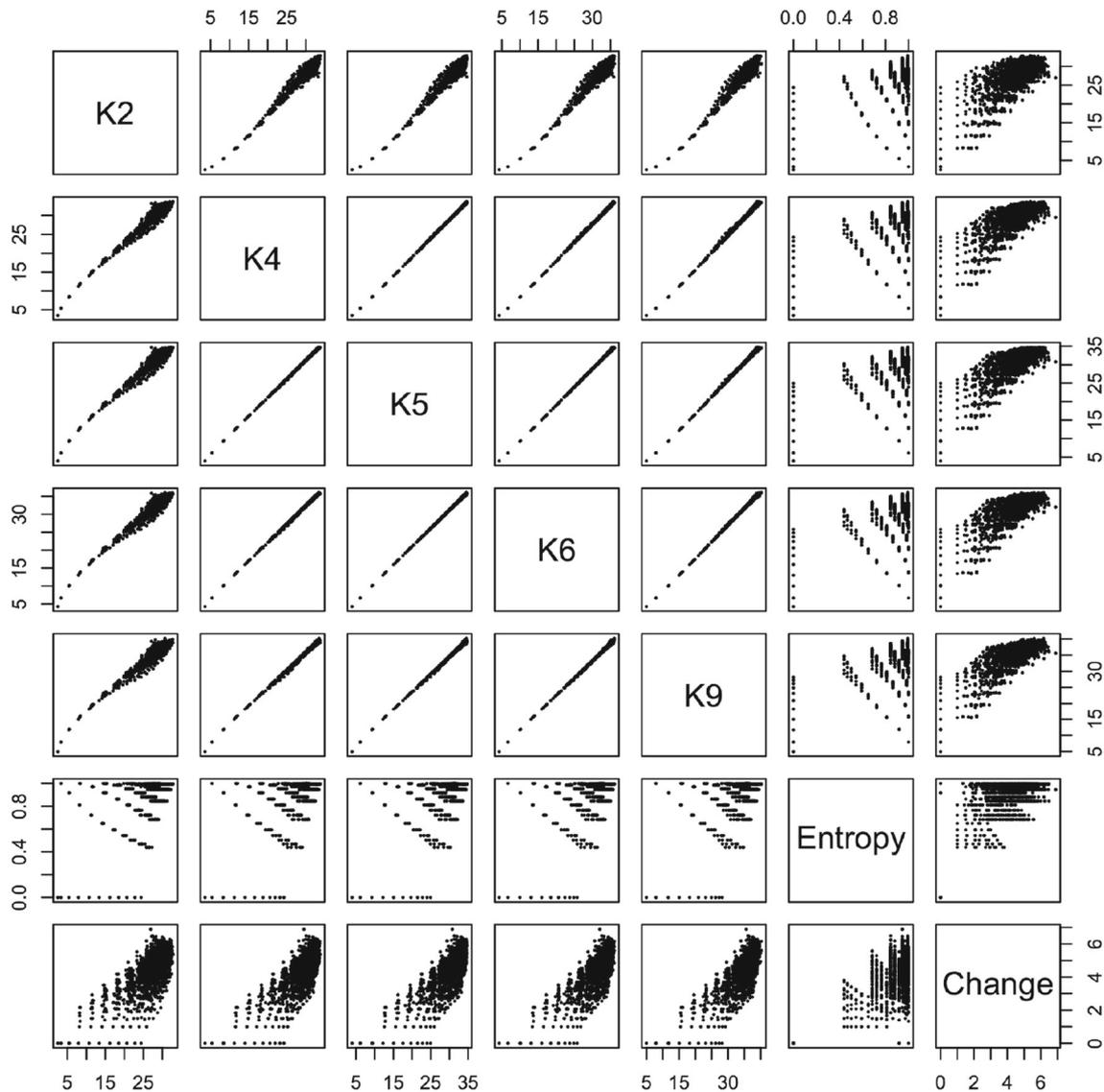
Fig. 2 Scatter plot showing the relation between measures of complexity on every binary string with length from 1 to 11. "Change" stands for change complexity

Any binary string can be thought of as an n-symbol string ($n \geq 2$) that happens to use only 2 symbols. For instance, "0101" could be produced by a machine that only uses 0s and 1s, but also by a machine that uses digits from 0 to 9. Hence "0101" may be viewed as a word based on the alphabet $\{0, 1\}$, but also based on $\{0, 1, 2, 3, 4\}$, etc. Therefore, the complexity of a binary string can be rated by K_2, but also by K_4 or K_5. We computed K_n (with $n \in \{2, 4, 5, 6, 9\}$), entropy and change complexity of all 2047 binary strings of length up to 11. Table 2 displays the resulting correlations and Fig. 2 shows the corresponding scatter plots. The different algorithmic complexity estimates through the coding theorem method are closely related, with correlations above 0.999 for K_4, K_5, K_6 and K_9. K_2 is less correlated to the others, but every correlation stands above 0.97. There is a

mild linear relation between ACSS and change complexity. Finally, entropy is only weakly linked to algorithmic and change complexity.

The correlation between different versions of ACSS (that is, K_n) may be partly explained by the length of the strings. Unlike entropy, ACSS is sensitive to the number of symbols in a string. This is not a weakness. On the contrary, if the complexity of a string is linked to the evidence it brings to the chance hypothesis, it should depend on length. Throwing a coin to determine if it is fair and getting HTHTTTHH is more convincing than getting just HT. Although both strings are balanced, the first one should be considered more complex because it affords more evidence that the coin is fair (and not, for instance, bound to alternate heads and tails). However, to control for the effect of length
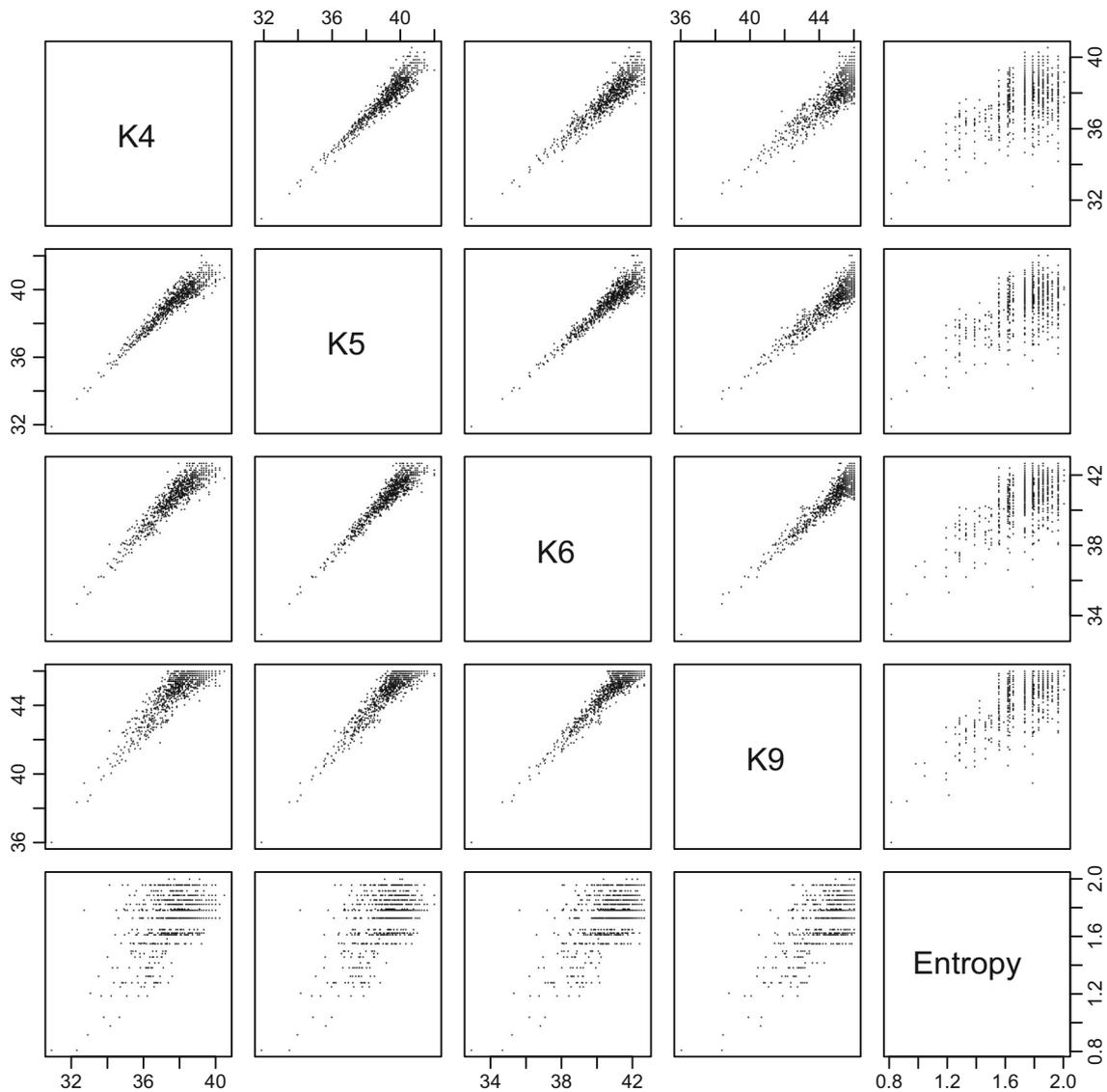
Fig. 3 Scatterplot matrix computed with 844 randomly chosen 12-character long 4-symbol strings

and extend our previous result to more complex strings, we picked up 844 random 4-symbol strings of length 12. Change complexity is not defined for non-binary sequences, but as Fig. 3 and Table 3 illustrate, the different ACSS measures are strongly correlated, and mildly correlated to entropy.

Table 3 Pearson's correlation between complexity measures, computed from 844 random 12-character long 4-symbol strings

|	K4	K5	K6	K9
K5	0.94			
K6	0.92	0.95		
K9	0.88	0.92	0.94	
Entropy	0.52	0.58	0.62	0.69

Applications

In this last section, we provide three illustrative applications of ACSS. The first two are short reports of new and illustrative experiments and the third one is a re-analysis of previously published data (Matthews, 2013). Although these experiments are presented to illustrate the use of ACSS, they also provide new insights into subjective probability and the perception of randomness. Note that all the data and analysis scripts for these applications are also part of `acss`.

Experiment 1: Humans are "better than chance"

Human pseudo-random sequential binary productions have been reported to be overly complex, in the sense that they

Fig. 4 Violin plot showing the distribution of complexity of human strings vs. every possible pattern of strings, with 4-symbol alphabet and length 10
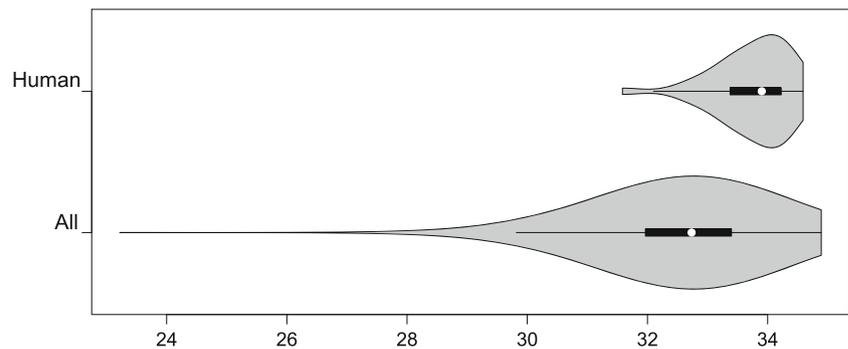
are more complex than the average complexity of truly random sequences (i.e., sequences of fixed length produced by repeatedly tossing a coin; (Gauvrit et al., 2013)). Here, we test the same effect with non-binary sequences based on 4 symbols. To replicate the analysis, type ?exp1 at the R prompt after loading acss and execute the examples.

Participants A sample of 34 healthy adults participated in this experiment. Ages ranged from 20 to 55 (mean = 37.65, SD = 7.98). Participants were recruited via e-mail and did not receive any compensation for their participation.

Methods Participants were asked to produce at their own pace a series of 10 symbols using "A", "B", "C", and "D" that would "look as random as possible, so that if someone else saw the sequence, she would believe it to be a truly random one". Participants submitted their responses via e-mail.

Results A one sample t-test showed that the mean complexity of the responses of participants is significantly larger that the mean complexity of all possible patterns of length 10 ($t(33) = 10.62$, $p < .0001$). The violin plot in Fig. 4 shows that human productions are more complex than random patterns because humans avoid low-complexity strings. On the other hand, human productions did not reach the highest possible values of complexity.

Discussion These results are consistent with the hypothesis that when participants try to behave randomly, they in fact tend to maximize the complexity of their responses, leading to overly complex sequences. However, whereas they succeed in avoiding low-complexity patterns, they cannot build the most complex strings.

Experiment 2: The threshold of complexity – a case study

Humans are sensitive to regularity and distinguish truly random series from deterministic ones (Yamada et al., 2013). More complex strings should be more likely to be considered random than simple ones. Here, we briefly address this

question through a binary forced choice task. We assume that there exists an individual threshold of complexity for which the probability that the individual identifies a string as random is .5. We estimated that threshold for one participant. The participant was a healthy adult male, 42 years old. The data and code are available by calling ?exp2.

Methods A series of 200 random strings of length 10 from an alphabet of 6 symbols, such as "6154256554", were generated with the R function sample(). For each string, the participant had to decide whether or not the sequence appeared random.

Results A logistic regression of the actual complexities of the strings (K_6) on the responses is displayed in Fig. 5. The results showed that more complex sequences were more likely to be considered random (slope = 1.9, $p < .0001$, corresponding to an odds ratio of 6.69). Furthermore, a complexity of 36.74 corresponded to the subjective probability of randomness 0.5 (i.e., the individual threshold was 36.74).

The span of local complexity

In a study of contextual effect in the perception of randomness, Matthews (2013, Experiment 1) showed participants series of binary strings of length 21. For each string, participants had to rate the sequence on a 6-point scale ranging from "definitely random" to "definitely not random". Results showed that participants were influenced by the context of presentation: sequences with medial alternation rate (AR) were considered highly random when they were intermixed with low AR, but as relatively non-random when intermixed with high AR sequences. In the following, we will analyze the data irrespective of the context or AR.

When individuals judge whether a short string of, for example, 3–6 characters, is random, they probably consider the complete sequence. For these cases, ACSS would be the right normative measure. When strings are longer, such as a length of 21, individuals probably cannot consider the complete sequence at once. Matthews (2013) and others (e.g.,
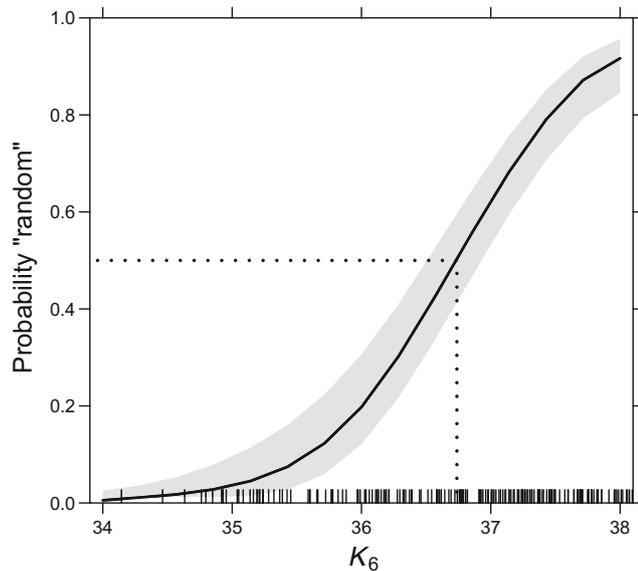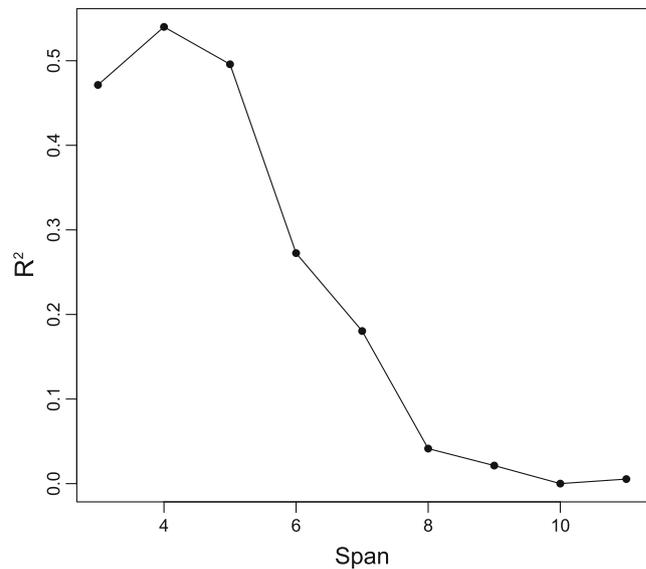
Fig. 5 Graphical display of the logistic regression with actual complexities (K_6) of 200 strings as independent variable and the observed responses (appears random or not) of one participant as dependent variable. The gray area depicts 95 %-confidence bands, the black dots at the bottom the 200 complexities. The dotted lines show the threshold where the perceived probability of randomness is 0.5

Fig. 6 R^2 between mean local complexity with span 3 to 11 and the subjective mean evaluation of randomness

(Hahn, 2014)) have hypothesized that in these cases individuals rely on the local complexity of the string. If this were true, the question remains as to how local the analysis is. To answer this, we will reanalyze Matthews' data.

For each string and each span ranging from 3 to 11, we first computed the mean local complexity of the string. For instance, the string "XXXXXXXOOOOXXXOOOOOOO" with span 11 gives a mean local complexity of 29.53. The same string has a mean local complexity of 11.22 with span 5.

```
> sapply(local_complexity
  ("XXXXXXXOOOOXXXOOOOOOO", 11, 2), mean)
XXXXXXXOOOOXXXOOOOOOO
              29.52912
> sapply(local_complexity
  ("XXXXXXXOOOOXXXOOOOOOO", 5, 2),  mean)
XXXXXXXOOOOXXXOOOOOOO
            11.21859
```

For each span, we then computed R^2 (the proportion of variance accounted for) between mean local complexity (a formal measure) and the mean randomness score given by the participants in Matthews' (2013), Experiment 1. Figure 6 shows that a span of 4 or 5 best describes the judgments with R^2 of 54 % and 50 %. Furthermore, R^2 decreases so fast that it amounts to less than 0.002 % when the span is set to 10. These results suggest that when asked to judge if a string is random, individuals rely on very local structural features of the strings, only considering subsequences of 4-5 symbols. This is very near

the suggested limit of the short term memory of 4 chunks (Cowan, 2001). Future research could build on this preliminary account to investigate the possible "span" of human observation in the face of possibly random serial data. The data and code for this application are available by calling `?matthews2013`.

Relationship to complexity based model selection

As mentioned in the beginning, complexity also plays an important role in modern approaches to model selection, specifically within the minimum description length framework (MDL; Grünwald (2007) and Myung et al. (2006)). Model selection in this context (see e.g. (Myung et al. In press)) refers to the process of selecting, among a set of candidate models, the model that strikes the best balance between goodness-of-fit (i.e., how well does the model describe the obtained data) and complexity (i.e., how well does the model describe all possible data or data in general). MDL provides a principled way of combining model fit with a quantification of model complexity that originated in information theory (Rissanen, 1989) and is, similar to algorithmic complexity, based on the notion of compressed code. The basic idea is that a model can be viewed as an algorithm that, in combination with a set of parameters, can produce a specific prediction. A model that describes the observed data well (i.e., provides a good fit) is a model that can compress the data well as it only requires a set of parameters and there are little residuals that need to be described in addition. Within this framework, the complexity of a model is the shortest possible code or algorithm that describes all

possible data pattern predicted by the model. The model selection index is the length of the concatenation of the code describing parameters and residuals and the code producing all possible data sets. As usual, the best model in terms of MDL is the model with the lowest model selection index.

The MDL approach differs from the complexity approach discussed in the current manuscript as it focuses on a specific set of models. To be selected as possible candidates, models usually need to satisfy other criteria such as providing explanatory value, need to be a priori plausible, and the parameters should be interpretable in terms of psychological processes (Myung et al., In press). In contrast, algorithmic complexity is concerned with finding the shortest possible description considering all possible models leaving those considerations aside. However, given that both approaches are couched within the same information theoretic framework, MDL converges towards algorithmic complexity if the set of candidate models becomes infinite (Wallace and Dowe, 1999). Furthermore, in the current manuscript we are only concerned with short strings, whereas even in compressed form data and the prediction space of a model are usually comparatively large.

Conclusion

Until the development of the coding theorem method (Soler-Toscano et al., 2014; Delahaye and Zenil, 2012), researchers interested in short random strings were constrained to use measures of complexity that focused on particular features of randomness. This has led to a number of (unsatisfactory) measures (Towse, 1998). Each of these previously used measures can be thought of as a way of performing a particular statistical test of randomness. In contrast, algorithmic complexity affords access to the ultimate measure of randomness, as the algorithmic complexity definition of randomness has been shown to be equivalent to defining a random sequence as one which would pass *every* computable test of randomness. We have computed an approximation of algorithmic complexity for short strings and made this approximation freely available in the R package acss.

Because human capacities are limited, it is unlikely that humans will be able to recognize every kind of deviation from randomness. Subjective randomness does not equal algorithmic complexity, as Griffiths and Tenenbaum (2004) remind us. Other measures of complexity will still be useful in describing the ways in which human pseudo-random behaviors or the subjective perception of randomness or complexity differ from objective randomness, as defined within the mathematical theory of randomness and advocated in this manuscript. But to achieve this goal of comparing subjective and objective randomness, we also need an objective and universal measure of randomness

(or complexity) based on a sound mathematical theory of randomness. Although the uncomputability of Kolmogorov complexity places some limitations on what is knowable about objective randomness, ACSS provides a sensible and practical approximation that researchers can use in real life. We are confident that ACSS will prove very useful as a normative measure of complexity in helping psychologists understand how human subjective randomness differs from "true" randomness as defined by algorithmic complexity.

References

Aksentijevic, A., & Gibson, K. (2012). Complexity equals change. *Cognitive Systems Research*, *15-16*, 1–16.

Audiffren, M., Tomporowski, P. D., & Zagrodnik, J. (2009). Acute aerobic exercise and information processing: Modulation of executive control in a random number generation task. *Acta Psychologica*, *132*(1), 85–95.

Baddeley, A. D., Thomson, N., & Buchanan, M. (1975). Word length and the structure of short-term memory. *Journal of Verbal Learning and Verbal Behavior*, *14*(6), 575–589.

Barbasz, J., Stettner, Z., Wierzchoń, M., Piotrowski, K. T., & Barbasz, A. (2008). How to estimate the randomness in random sequence generation tasks? *Polish Psychological Bulletin*, *39*(1), 42 – 46.

Bédard, M. J., Joyal, C. C., Godbout, L., & Chantal, S. (2009). Executive functions and the obsessive-compulsive disorder: On the importance of subclinical symptoms and other concomitant factors. *Archives of Clinical Neuropsychology*, *24*(6), 585–598.

Bianchi, A. M., & Mendez, M. O. (2013). Methods for heart rate variability analysis during sleep. In *Engineering in Medicine and Biology Society (embc), 2013 35th Annual International Conference of the ieee* (pp. 6579–6582).

Boon, J. P., Casti, J., & Taylor, R. P. (2011). Artistic forms and complexity. *Nonlinear Dynamics-Psychology and Life Sciences*, *15*(2), 265.

Brandouy, O., Delahaye, J. P., Ma, L., & Zenil, H. (2012). Algorithmic complexity of financial motions. *Research in International Business and Finance*, *30*(C), 336–347.

Brown, R., & Marsden, C. (1990). Cognitive function in parkinson's disease: From description to theory. *Trends in Neurosciences*, *13*(1), 21–29.

Calude, C. (2002). *Information and randomness. An algorithmic perspective* (2nd, revised and extended). Berlin Heidelberg: Springer.

Cardaci, M., Di Gesu, V., Petrou, M., & Tabacchi, M. E. (2009). Attentional vs computational complexity measures in observing paintings. *Spatial Vision*, *22*(3), 195–209.

Chaitin, G. (1966). On the length of programs for computing finite binary sequences. *Journal of the ACM*, *13*(4), 547–569.

Chaitin, G. (2004). *Algorithmic information theory* (Vol. 1). Cambridge: Cambridge University Press.

Chater, N. (1996). Reconciling simplicity and likelihood principles in perceptual organization. *Psychological Review*, *103*(3), 566–581.

Chater, N., & Vitányi, P. (2003). Simplicity: a unifying principle in cognitive science? *Trends in Cognitive Sciences*, *7*(1), 19–22.

Cilibrasi, R., & Vitányi, P. (2005). Clustering by compression. *Information Theory, IEEE Transactions on*, *51*(4), 1523–1545.

Cilibrasi, R., & Vitányi, P. (2007). The google similarity distance. *Knowledge and Data Engineering, IEEE Transactions on*, *19*(3), 370–383.

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*(1), 87–114.

Crova, C., Struzzolino, I., Marchetti, R., Masci, I., Vannozzi, G., & Forte, R. (2013). Cognitively challenging physical activity benefits executive function in overweight children. *Journal of Sports Sciences, ahead-of-print*, 1–11.

Curci, A., Lanciano, T., Soleti, E., & Rimé, B. (2013). Negative emotional experiences arouse rumination and affect working memory capacity. *Emotion*, *13*(5), 867–880.

Delahaye, J. P., & Zenil, H. (2012). Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Applied Mathematics and Computation*, *219*(1), 63–77.

Downey, R. R. G., & Hirschfeldt, D. R. (2008). *Algorithmic randomness and complexity*. Berlin Heidelberg: Springer.

Elzinga, C. H. (2010). Complexity of categorical time series. *Sociological Methods & Research*, *38*(3), 463–481.

Feldman, J. (2000). Minimization of boolean complexity in human concept learning. *Nature*, *407*(6804), 630–633.

Feldman, J. (2003). A catalog of boolean concepts. *Journal of Mathematical Psychology*, *47*(1), 75–189.

Feldman, J. (2006). An algebra of human concept learning. *Journal of Mathematical Psychology*, *50*(4), 339–368.

Fernández, A., Quintero, J., Hornero, R., Zuluaga, P., Navas, M., & Gómez, C. (2009). Complexity analysis of spontaneous brain activity in attention-deficit/hyperactivity disorder: Diagnostic implications. *Biological Psychiatry*, *65*(7), 571–577.

Fernández, A., Ríos-Lago, M., Abásolo, D., Hornero, R., Álvarez-Linera, J., & Paul, N. (2011). The correlation between white-matter microstructure and the complexity of spontaneous brain activity: A difussion tensor imaging-meg study. *Neuroimage*, *57*(4), 1300–1307.

Fernández, A., Zuluaga, P., Abásolo, D., Gómez, C., Serra, A., & Méndez, M. A. (2012). Brain oscillatory complexity across the life span. *Clinical Neurophysiology*, *123*(11), 2154–2162.

Fournier, K. A., Amano, S., Radonovich, K. J., Bleser, T. M., & Hass, C. J. (2013). Decreased dynamical complexity during quiet stance in children with autism spectrum disorders. *Gait & Posture*.

Free Software Foundation (2007). *GNU general public license*. Retrieved from http://www.gnu.org/licenses/gpl.html

Gauvrit, N., Soler-Toscano, F., & Zenil, H. (2014). Natural scene statistics mediate the perception of image complexity. *Visual Cognition*, *22*(8), 1084–1091.

Gauvrit, N., Zenil, H., Delahaye, J. P., & Soler-Toscano, F. (2013). Algorithmic complexity for short binary strings applied to psychology: A primer. *Behavior Research Methods*, *46*(3), 732–744.

Griffiths, T. L., & Tenenbaum, J. B. (2003). Probability, algorithmic complexity, and subjective randomness. In R. Alterman & D. Kirsch (Eds.)*, Proceedings of the 25th annual conference of the cognitive science society* (pp. 480–485). Mahwah, NJ: Erlbaum.

Griffiths, T. L., & Tenenbaum, J. B. (2004). From algorithmic to sub- jective randomness. In S. Thrun, L. K. Saul & B. Schölkopf (Eds.)*, Advances in neural information processing systems* (Vol. 16 pp. 953–960). Cambridge, MA: MIT Press.

Gruber, H. (2010). *On the descriptional and algorithmic complexity of regular languages*. Justus Liebig University Giessen.

Grünwald, P. D. (2007). *The minimum description length principle*. Cambridge: MIT Press.

Hahn, T., Dresler, T., Ehlis, A. C., Pyka, M., Dieler, A. C., & Saathoff, C. (2012). Randomness of resting-state brain oscillations encodes gray's personality trait. *Neuroimage*, *59*(2), 1842–1845.

Hahn, U. (2014). Experiential limitation in judgment and decision. *Topics in Cognitive Science*, *6*(2), 229–244.

Hahn, U., Chater, N., & Richardson, L. B. (2003). Similarity as transformation. *Cognition*, *87*(1), 1–32.

Hahn, U., & Warren, P.A. (2009). Perceptions of randomness: why three heads are better than four. *Psychological Review*, *116*(2), 454–461.

Heuer, H., Kohlisch, O., & Klein, W. (2005). The effects of total sleep deprivation on the generation of random sequences of key-presses, numbers and nouns. *The Quarterly Journal of Experimental Psychology A: Human Experimental Psychology*, *58A*(2), 275–307.

Hsu, A. S., Griffiths, T. L., & Schreiber, E. (2010). Subjective randomness and natural scene statistics. *Psychonomic Bulletin & Review*, *17*(5), 624–629.

Jones, O., Maillardet, R., & Robinson, A. (2009). *Introduction to scientific programming and simulation using R*. Boca Raton, FL: Chapman & Hall/CRC.

Kahneman, D., Slovic, P., & Tversky, A. (1982). *Judgment under uncertainty: Heuristics and biases*. Cambridge: Cambridge University Press.

Kass, R. E., & Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, *90*(430), 773–795. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/01621459.1995.10476572

Kellen, D., Klauer, K.C., & Bröder, A. (2013). Recognition memory models and binary-response ROCs: A comparison by minimum description length. *Psychonomic Bulletin & Review*, *20*(4), 693–719.

Koike, S., Takizawa, R., Nishimura, Y., Marumo, K., Kinou, M., & Kawakubo, Y. (2011). Association between severe dorsolateral prefrontal dysfunction during random number generation and earlier onset in schizophrenia. *Clinical Neurophysiology*, *122*(8), 1533–1540.

Kolmogorov, A. (1965). Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, *1*(1).

Lai, M. C., Lombardo, M. V., Chakrabarti, B., Sadek, S. A., Pasco, G., & Wheelwright, S. J. (2010). A shift to randomness of brain oscillations in people with autism. *Biological Psychiatry*, *68*(12), 1092–1099.

Levin, L. A. (1974). Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, *10*(3), 30–35.

Li, M., & Vitányi, P. (2008). *An introduction to kolmogorov complexity and its applications*. Berlin Heidelberg: Springer.

Loetscher, T., & Brugger, P. (2009). Random number generation in neglect patients reveals enhanced response stereotypy, but no neglect in number space. *Neuropsychologia*, *47*(1), 276 – 279.

Machado, B., Miranda, T., Morya, E., Amaro Jr, E., & Sameshima, K. (2010). P24-23 algorithmic complexity measure of EEG for staging brain state. *Clinical Neurophysiology*, *121*, S249–S250.

Maes, J. H., Vissers, C. T., Egger, J. I., & Eling, P. A. (2012). On the relationship between autistic traits and executive functioning in a non-clinical Dutch student population. *Autism*, *17*(4), 379–389.

Maindonald, J., & Braun, W. J. (2010). *Data analysis and graphics using R: An example-based approach,* 3rd Edn. Cambridge: Cambridge University Press.

Manktelow, K. I., & Over, D. E. (1993). Rationality: Psychological and philosophical perspectives. Taylor & Frances/Routledge.

Martin-Löf, P. (1966). The definition of random sequences. *Information and control*, *9*(6), 602–619.

Mathy, F., & Feldman, J. (2012). What's magic about magic numbers? Chunking and data compression in short-term memory. *Cognition*, *122*(3), 346–362.

Matloff, N. (2011). *The art of R programming: A tour of statistical software design,* 1st Edn. San Francisco: No Starch Press.

Matthews, W. (2013). Relatively random: Context effects on perceived randomness and predicted outcomes. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *39*(5), 1642-1648.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, *63*(2), 81–97.

Myung, J. I., Cavagnaro, D. R., & Pitt, M. A. (In press). New handbook of mathematical psychology, vol. 1: Measurement and methodology. In W. H. Batchelder, H. Colonius, E. Dzhafarov & J. I. Myung (Eds.), *(chap. Model evaluation and selection)*. Cambridge: Cambridge University Press.

Myung, J. I., Navarro, D. J., & Pitt, M. A. (2006). Model selection by normalized maximum likelihood. *Journal of Mathematical Psychology*, *50*(2), 167–179.

Naranan, S. (2011). Historical linguistics and evolutionary genetics. Based on symbol frequencies in tamil texts and dna sequences. *Journal of Quantitative Linguistics*, *18*(4), 337–358.

Nies, A. (2009). *Computability and randomness,* Vol. 51. London: Oxford University Press.

Over, D. E. (2009). New paradigm psychology of reasoning. *Thinking & Reasoning*, *15*(4), 431–438.

Pearson, D. G., & Sawyer, T. (2011). Effects of dual task interference on memory intrusions for affective images. *International Journal of Cognitive Therapy*, *4*(2), 122–133.

Proios, H., Asaridou, S. S., & Brugger, P. (2008). Random number generation in patients with aphasia: A test of executive functions. *Acta Neuropsychologica*, *6*, 157–168.

Pureza, J. R., Gonçalves, H. A., Branco, L., Grassi-Oliveira, R., & Fonseca, R. P. (2013). Executive functions in late childhood: Age differences among groups. *Psychology & Neuroscience*, *6*(1), 79–88.

R Core Team (2014). R: A language and environment for statistical computing [Vienna, Austria]. Retrieved from http://www.R-project.org/

Rado, T. (1962). On non-computable functions. *Bell System Technical Journal*, *41*, 877-884.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry theory*. World Scientific Publishing Co., Inc.

Ryabko, B., Reznikova, Z., Druzyaka, A., & Panteleeva, S. (2013). Using ideas of Kolmogorov complexity for studying biological texts. *Theory of Computing Systems*, *52*(1), 133–147.

Scafetta, N., Marchi, D., & West, B. J. (2009). Understanding the complexity of human gait dynamics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *19*(2), 026108.

Schnorr, C. P. (1973). Process complexity and effective random tests. *Journal of Computer and System Sciences*, *7*(4), 376–388.

Schulter, G., Mittenecker, E., & Papousek, I. (2010). A computer program for testing and analyzing random generation behavior in normal and clinical samples: The mittenecker pointing test. *Behavior Research Methods*, *42*, 333–341.

Scibinetti, P., Tocci, N., & Pesce, C. (2011). Motor creativity and creative thinking in children: The diverging role of inhibition. *Creativity Research Journal*, *23*(3), 262–272.

Shannon, C. E. (1948). A mathematical theory of communication, part I. *Bell Systems Technical Journal*, *27*, 379–423.

Sokunbi, M. O., Fung, W., Sawlani, V., Choppin, S., Linden, D. E., & Thome, J. (2013). Resting state fMRI entropy probes complexity of brain activity in adults with ADHD. *Psychiatry Research: Neuroimaging*, *214*(3), 341–348.

Soler-Toscano, F., Zenil, H., Delahaye, J. P., & Gauvrit, N. (2013). Correspondence and independence of numerical evaluations of algorithmic information measures. *Computability*, *2*(2), 125–140.

Soler-Toscano, F., Zenil, H., Delahaye, J. P., & Gauvrit, N. (2014). Calculating Kolmogorov complexity from the output frequency distributions of small turing machines. *PLOS One*, *9*, e96223.

Solomonoff, R. J. (1964a). A formal theory of inductive inference. Part I. *Information and Control*, *7*(1), 1–22.

Solomonoff, R. J. (1964b). A formal theory of inductive inference. Part II. *Information and Control*, *7*(2), 224–254.

Takahashi, T. (2013). Complexity of spontaneous brain activity in mental disorders. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, *45*, 258–266.

Taufemback, C., Giglio, R., & Da Silva, S. (2011). Algorithmic complexity theory detects decreases in the relative efficiency of stock markets in the aftermath of the 2008 financial crisis. *Economics Bulletin*, *31*(2), 1631–1647.

Towse, J. N. (1998). Analyzing human random generation behavior: A review of methods used and a computer program for describing performance. *Behavior Research Methods*, *30*(4), 583–591.

Towse, J. N., & Cheshire, A. (2007). Random number generation and working memory. *European Journal of Cognitive Psychology*, *19*(3), 374–394.

Tversky, A., & Kahneman, D. (1974). Judgment under uncertainty: Heuristics and biases. *Science*, *185*(4157), 1124–1131.

Wagenaar, W. A. (1970). Subjective randomness and the capacity to generate information. *Acta Psychologica*, *33*, 233–242.

Wallace, C. S., & Dowe, D. L. (1999). Minimum message length and Kolmogorov complexity. *The Computer Journal*, *42*(4), 270–283.

Watanabe, T., Cellucci, C., Kohegyi, E., Bashore, T., Josiassen, R., & Greenbaun, N. (2003). The algorithmic complexity of multichannel EEGs is sensitive to changes in behavior. *Psychophysiology*, *40*(1), 77–97.

Wiegersma, S. (1984). High-speed sequential vocal response production. *Perceptual and Motor Skills*, *59*, 43–50.

Wilder, J., Feldman, J., & Singh, M. (2011). Contour complexity and contour detectability. *Journal of Vision*, *11*(11), 1044.

Williams, J. J., & Griffiths, T. L. (2013). Why are people bad at detecting randomness? A statistical argument. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *39*(5), 1473–1490.

Yagil, G. (2009). The structural complexity of dna templatesimplications on cellular complexity. *Journal of Theoretical Biology*, *259*(3), 621–627.

Yamada, Y., Kawabe, T., & Miyazaki, M. (2013). Pattern randomness aftereffect. *Scientific Reports*, *3*.

Yang, A. C., & Tsai, S. J. (2012). Is mental illness complex? From behavior to brain. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, *45*, 253–257.

Zabelina, D. L., Robinson, M. D., Council, J. R., & Bresin, K. (2012). Patterning and nonpatterning in creative cognition: Insights from performance in a random number generation task. *Psychology of Aesthetics, Creativity, and the Arts*, *6*(2), 137–145.

Zenil, H. (2011a). *Randomness through computation: Some answers, more questions*. World Scientific.

Zenil, H. (2011b). Une approche expérimentale de la théorie algorith-mique de la complexité Une approche expérimentale de la théorie algorithmique de la complexité. Universidad de Buenos Aires.

Zenil, H., & Delahaye, J. P. (2010). On the algorithmic nature of the world. In G. Dodig-Crnkovic & M. Burgin (Eds.), *Information and computation* (pp. 477–496). World Scientific.

Zenil, H., & Delahaye, J. P. (2011). An algorithmic information the-oretic approach to the behaviour of financial markets. *Journal of Economic Surveys*, *25*(3), 431–463.

Zenil, H., Soler-Toscano, F., Delahaye, J., & Gauvrit, N. (2012). Two-dimensional Kolmogorov complexity and validation of the coding theorem method by compressibility. CoRR, arXiv:1212.6745