CDMTCS
Research
Report
Series

Randomness and One-way Functions

José Manuel Agüero Trejo and Cristian S. Calude

School of Computer Science, University of Auckland, New Zealand

CDMTCS-581
November 2024

Centre for Discrete Mathematics and
Theoretical Computer Science

Randomness and One-way Functions

José Manuel AGÜERO TREJO[1] and Cristian S. CALUDE[2, *]

[1]School of Computer Science, University of Auckland, New Zealand
[2]School of Computer Science, University of Auckland, New Zealand
Email:jagu688@aucklanduni.ac.nz, cristian@cs.auckland.ac.nz*
* Corresponding author

Abstract. This paper proposes a method using high-quality randomness for inverting a class of one-way functions in a pre-given time. Experimental results suggest that the method is better than the lexicographic search.

Key-words: algorithmic random string; program-size complexity, Borel normality, randomised search, experiment

1. Introduction

Primality testing is "theoretically easy" because, in 2002, M. Agrawal, N. Kayal, and N. Saxena discovered a deterministic polynomial time solution for it [1]; the best primality test works in $O((\log n)^6)$. To date, deterministic primality tests are impractical, so in many applications, probabilistic tests of primality, as Solovay-Strassen algorithm [2, 3], are used instead. These algorithms work with the tested number and other numbers chosen randomly from a sample space. Randomised tests never report a prime number as a composite. However, a composite number can be reported as prime with a probability of error, which can be made arbitrarily small by repeating the test with several independently chosen random values.

Inspired by this analogy, we explore the use of high-quality random numbers in tasks involving functions believed to be one-way functions. In particular, we look at the task of finding inputs (for this type of function) that produce output strings sharing at least one pattern of interest. Moreover, we have a time restriction for completing this task. It is commonly assumed that one can do no better than trying all possible input strings (in lexicographic order) until the desired result is obtained. In this paper, we use high-quality randomness to reduce the search space, which provides a more efficient algorithm.

The article is structured as follows. In the next section we present a few results from Algorithmic Information Theory, [4]; Section 3. is dedicated to Borel normality, a computable symptom of randomness; the main Section 4. is dedicated to the randomised search; Section 5. presents one-way functions; the last Section 6. described an experiment which illustrates the superiority of the randomise method compared with the lexicographic one.

2. Program-size complexity

Consider the alphabet $A_b = \{0, 1, 2, \ldots, b-1\}$ for $b \geq 2$. We work with strings $x \in A_b^*$ and Turing Machines (TMs) M to perform computations [5]. The length of the string x is denoted by $|x|_b$ or simply $|x|$. A *description* of a pair of two strings, $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_m$ is the string $\langle x, y\rangle = x_1 0 \ldots x_{n-1} 0 x_n 1 y$. The length of $\langle x, y\rangle$ is $|\langle x, y\rangle| = 2|x| + |y|$. Two computable functions can retrieve the components x, y of $\langle x, y\rangle$.

Strings are described by TM computations. A *description* of the string x is given by a pair (M, w) where M is a string representing a TM and w is an input string to M such that $M(w) = x$. The *length of the description* of the string x given by M and w is $|\langle M, w\rangle| = 2|\langle M\rangle| + |w|$.

A *minimal description* of x, written $d(x)$, is the shortest description $\langle M, w\rangle$ for x. *The Kolmogorov-Chaitin complexity* of x, written $K(x)$, is $K(x) = |d(x)|$.

The complexity K is incomputable and grows linearly: there exists a positive integer c such that every string $x \in A_b^*$, $K(x) \leq |x| + c$, that is, $K(x)$ is not much larger than the x. Of course, the complexity of *some* strings can be much shorter than the lengths of the strings. For example,

$$A = 0101010101010101010101010101010101010101010101010101010101010101,$$

$$B = 0101101010100101010110101010010101011010101001010101101010100101.$$

appears to be shorter than their lengths.

Is this case for all strings? Can the strings generated with a 3D QRNG [6] using the method developed in [7] be compressed:

$$A = 01101100101010010011001001011111100011111010110011011000010100001,$$

$$B = 0011101101010001110010111110100111100100001000000101001001011011.$$

Each description is a string in A_b^*, so the number of descriptions of length less than n is at most the sum of the numbers of strings length $0, 1, \ldots, n-1$: $1 + b + b^2 + \cdots + b^{n-1} = (b^n - 1)/(b-1)$, which is less than b^n, the number of strings of length n; so at least one string of length n has to have the complexity $K(x) \geq n = |x|$.

We say that a string x is c-compressible ($1 \leq c < |x|$) if $K(x) \leq |x| - c$. If x is not c-compressible, then x is c-incompressible. Incompressible strings of every length $c > 0$ exist:

$$\#\{x \in A_b^n \mid K(x) > |x| - c\} = b^n - \#\{x \in A_b^n \mid K(x) > |x| - c\} \geq b^n - \frac{b^{n-c-1}}{b-1} > 0.$$

Which set is larger: $\{x \in A_b^n \mid K(x) > |x| - c\}$ or $\{x \in A_b^n \mid K(x) \leq |x| - c\}$?

$$\begin{aligned}\#\{x \in A_b^n \mid K(x) > |x| - c\} &= b^n - \#\{x \in A_b^n \mid K(x) \leq n - c\} \\ &= b^n - (1 + 2 + \cdots + b^{n-c}) \\ &> \#\{x \in A_b^n \mid K(x) \leq |x| - c\}.\end{aligned}$$

The probability that a string $x \in A_b^n$ is c-compressible is

$$\frac{\#\{x \in A_b^n \mid K(x) \leq |x| - c\}}{b^n} < \frac{1}{(b-1)b^{1-c}}, \tag{1}$$

while the probability that a string $x \in A_b^n$ is c-incompressible is

$$\frac{\#\{x \in A_b^n \mid K(x) \leq |x| - c\}}{b^n} < 1 - \frac{1}{(b-1)b^{1-c}}.$$

Hence, these probabilities are not equal to 2^{-n} as the classical probability asserts.

Take now $c = n/2$, a mild compression degree and $b = 2$, i.e. strings are binary. Then we have:

$$\begin{aligned}\#\{|x| = n \mid K(x) \leq n - n/2\} \cdot 2^{-n} &= \#\{|x| = n \mid K(x) \leq n/2\} \cdot 2^{-n} \\ &\leq 2^{1-n/2} \longrightarrow_{n\to\infty} 0,\end{aligned}$$

while the probability that a string x of length n is not $n/2$-compressible is

$$\begin{aligned}\#\{|x| = n \mid K(x) \geq n - n/2\} \cdot 2^{-n} &= \#\{|x| = nemphasize \geq n/2\} \cdot 2^{-n} \\ &\geq 1 - 2^{1-n/2} \longrightarrow_{n\to\infty} 1.\end{aligned}$$

3. Borel normality

Fix an integer $m > 1$ and consider the alphabet $A_b^m = \{a_1, \ldots, a_{b^m}\}$ of all strings $x \in A_b^*$ with $|x|_b = m$, ordered lexicographically. A string $x \in A_b^*$ will be denoted by x^m when we emphasise that it belongs to $(A_b^m)^*$. By A_b^ω we denote the set of all infinite sequences $\mathbf{x} = x_1 x_2 \cdots$ with $x_i \in A_b^*$.

Take for example, for $A_2 = \{0, 1\}, m = 2, A_2^2 = \{00, 01, 10, 11\}$; the string $x = 10110100 \in A_2^*$ will be denoted by $x^2 = (10)(11)(01)(00)$ when considered in A_2^2. Clearly, $|x|_2 = 8$ and $|x^2|_4 = 4$. In the same way a sequence $\mathbf{x} \in A_b^\omega$ will be written as $\mathbf{x}^m$ when considered in $(A_b^m)^\omega$.

Let $N_i(x)$ be the number of occurrences of $i \in A_b$ in the string $x \in A_b^*$ and for every $u \in A_b^m$ let $N_u^m(x^m)$ be the number of occurrences of u in the string $x^m \in (A_b^m)^*$. In the example above $N_0^1(x) = N_1^1(x) = 4$ and $N_{11}^2(x^2) = N_{10}^2(x^2) = N_{01}^2(x^2) = N_{00}^2 = 1$.

For $\mathbf{x} \in A_b^\omega$ and $n \geq 1$, $\mathbf{x}(n) = x_1 x_2 \ldots x_n \in A_b^*$. The sequence x is called *Borel-m normal* $(m \geq 1)$ if for every $u \in (A_b^m)^*$ one has:

$$\lim_{n\to\infty} \frac{N_u^m(\mathbf{x}^m(\lfloor \frac{n}{m} \rfloor))}{\lfloor \frac{n}{m} \rfloor} = \frac{1}{b^m}.$$

The sequence $\mathbf{x} \in A_b^\omega$ is called *Borel normal* if it is Borel-m normal, for every natural number. In particular, a sequence x is Borel-1 normal when for every $a \in A_b$ we have:

$$\lim_{n\to\infty} \frac{N_a(\mathbf{x}(n))}{n} = \frac{1}{b}.$$

This definition is the "conventional" notion of normality.

The Borel Law of Large Numbers [8] shows that with probability one every sequence is Borel normal. This result is asymptotic; in a practical scenario, we do not have infinite bit-strings; hence to analyse this property for prefixes of an arbitrary bit-string we use the finite version of Borel normality [9].

For every $\varepsilon > 0$ and integer $m > 1$ we say that a string $x \in A_2^*$ is *Borel normal with accuracy* (m, ε) if

$$\left| \frac{N_u^m(x^m(\lfloor \frac{|x|_2}{m} \rfloor))}{\lfloor \frac{|x|_2}{m} \rfloor} - 2^{-m} \right| \leq \varepsilon, \tag{2}$$

for each $u \in A_2^m$ and $1 \leq m \leq \log_2 \log_2 |x|_2$.

Almost all algorithmic random strings of any length are Borel normal with these accuracies [4, 9]. Furthermore, *if all prefixes of a bit sequence are Borel normal, then the sequence itself is also Borel normal.*

When only finitely many bits of a sequence can be computed, we can test its prefixes for Borel normality corresponding to the allowable values of m, [10–12]).

Borel normality of prefixes is a computable property which does not guarantee that a string is incomputable. For example, for any b, the computable Champernowne number $0.012345678910\ldots$ is normal in base b.

4. Randomised search

In this section we present the Randomised search.

Let $A = \{0,1\}^n$ and $B = \{0,1\}^m$ for fixed $m, n \in \mathbb{N}$ and $n \geq m \geq 32$. Let $f : A \to B$ be a computable function with a normal distribution of outputs, that is, for an every $x \in A$, $f(x)$ is Borel-1 normal. *The task is to find a string $a \in A$ such that $f(a) \in T$ within a given number of calls to f.*

Let $x = x_1 x_2 \ldots x_k$ be a binary string. A pattern $\mathbf{p} = p_1^{i_1} \ldots p_l^{i_l}$ consists of $p_j \in \{0,1\}$ and a set of indices $I(\mathbf{p}) = \{i_1, \ldots, i_l | 0 < i_j \leq k\}$. The pattern $\mathbf{p}$ appears in the string x if for every index $i_j \in I(\mathbf{p})$ we have that $p_j^{i_j} = x_{i_j}$ For example, the pattern $\mathbf{p} = 1^1 0^2 0^4$ appears in each element of the set $\{\mathbf{10}0\mathbf{0}0, \mathbf{10}0\mathbf{0}1, \mathbf{10}1\mathbf{0}0, \mathbf{10}1\mathbf{0}1\}$.

Let $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_r\}$ be a set of r patterns and let $T \subset B$ be such that every string in T has at least one pattern in $\mathbf{P}$. Let $L = \{l_1, \ldots l_r\}$ be the set of base two expansions of the lengths of the patterns in $\mathbf{P}$. For example, for $\mathbf{P} = \{1^1 0^2 0^4, 0^2 0^3\}$ we have that $|I(\mathbf{p_1})| = 3$ and $|I(\mathbf{p_2})| = 2$, so $L = \{11, 10\}$

We say that the function $\mathcal{P} : B \to B$ applies the pattern $\mathbf{p} = p_1^{i_1} \ldots p_l^{i_l}$ on the string x if it replaces its characters at positions $i_1, i_2, \ldots, i_l$ by $p_1, \ldots, p_l$. For example, for $x = 10110100$ and $\mathbf{p} = 1^1 0^3 0^4$ we have $\mathcal{P}(x) = 1\mathbf{1}1\mathbf{00}0100$.

Let $\mathbf{p} = p_1^{i_1} \ldots p_r^{i_r}$ be the longest contiguous pattern in $\mathbf{P}$. That is, the longest pattern $\mathbf{p} \in \mathbf{P}$ such that, for $i_j \in I(\mathbf{p})$ with $1 < j \leq r$, we have that $|i_j - i_{j-1}| = 1$. We do the following if there are no contiguous patterns in $\mathbf{P}$. Let r be the length of the longest pattern in $\mathbf{P}$. Now, for every pattern $\mathbf{p} \in \mathbf{P}$, we look at the longest contiguous sub-pattern in $\mathbf{p}$ and pick one of the largest.

For example, in the pattern $\mathbf{0^{80} 1^{81} 0^{82} 1^{83}} 0^{100} 0^{101}$, the bold elements correspond to the longest contiguous sub-pattern.

We then concatenate it to itself until it has length r and discard any leftover bits. We set $\mathbf{p_c}$ to be the resulting string with corresponding index set $I(\mathbf{p}_c) = \{1, 2, 3 \ldots, r\}$, to obtain a contiguous pattern. For example, for

$$\mathbf{P} = \{1^1 1^{10} 0^{20} 1^{30} 0^{40} 1^{50}, 0^{80} 1^{81} 0^{82} 1^{83} 0^{100} 0^{101}\},$$

the longest pattern is $\mathbf{p}_1$ with length $r = 6$ but it is not contiguous. So we take the longest contiguous sub-pattern instead, which in this example is $0^{80} 1^{81} 0^{82} 1^{83}$ in $\mathbf{p}_2$. Thus we set $\mathbf{p_c} := 0^1 1^2 0^3 1^4 0^5 1^6$.

We now present in detail the Randomised search algorithm. First, we set $\mathbf{p}$ to be the longest contiguous pattern $\mathbf{p_c}$ as defined above. Then, as a measure of time, we set a limit $t \in \mathbb{N}$ to the number of calls to f. Finally, in some cases we may have access to a subset $T' \subset T$ of strings generated through the function f. If this is not the case, we set $T' = \emptyset$.

Randomised search $(n, f, T, T', L, \mathbf{p}, t)$:

1. Use the lengths of strings in T to estimate the largest value k for which we conduct a Borel-k normality test. If $T' \neq \emptyset$, test the Borel-k normality of the strings in T'(this process is presented in detail in [12].) Let λ be the average Borel-k normality of those strings.
2. Generate a string a of length n using a genuine random number generator (GRNG[1]) and set $j := 1$.
3. If $f(a) \in T$, stop and return a.
   Else, find the first string $l \in L$ that appears in a after the index j.
   If no such l is found, go to Step 2.
4. Let j' be the index of the first character following l.
   Set $\mathbf{p} := p_1^{(i_1 - i_1) + j} \ldots, p_r^{(i_r - i_1) + j}$, $a := \mathcal{P}(a)$ and $j := j'$.
5. Repeat Step 4 until one of the following occurs:

[1] That is, a random number generator provably better than any pseudo-random number generator, e.g. the 3D quantum number generator [13].

- $f(a) \in T$: stop and return a.
- Testing a for Borel-k normality results in a higher value than λ, if λ is known.
- More than k repetitions have been performed.
- The limit t has been reached: stop.

6. Go to Step 2.

For example, consider the cryptographic hash function SHA-512, [14], a computable function with Borel-1 normal 512-bit long outputs. We fix the length of the inputs to $n = 600$ bits. Suppose we are interested in finding a pattern consisting of 5 or 7 alternating 1's and 0's in the middle of the outputs strings, that is, $\mathbf{P} = \{\mathbf{p_1}, \mathbf{p_2}\}$ with

$$\mathbf{p_1} = 1^{128}0^{129}1^{130}0^{131}1^{132}, \mathbf{p_2} = 0^{128}1^{129}0^{130}1^{131}0^{132}1^{133}0^{134}.$$

The lengths of these patterns are 5 and 7, respectively, so we have that $L = \{101, 111\}$. Finally, since the outputs are 512 bits long and $\log_2 \log_2(512) = 3.17$, we can test for Borel-$k$ normality for $1 \leq k \leq 3$.

We can now start the Randomised search: Step 1: We test for Borel-3 normality; in Step 2 we use a GRNG to generate a string of 600 bits and set our index j to 1; in Step 3 with the string $a = 10001010110111010\ldots$ with $|a| = 600$ such that $f(a)$ does not have either of the patterns in $\mathbf{P}$, that is, $f(a) \notin T$, we find the first occurrence of any $l \in L$. Note that l_1 with corresponding pattern $\mathbf{p}_1$ occurs at the beginning of the string $a = 1000\mathbf{101}0110111010\ldots$ ending at position 7. In Step 4 we set $j' = 8$. Then, we set

$$\mathbf{p} := 1^1 0^2 1^3 0^4 1^5$$

and $a := \mathcal{P}(a) = \mathbf{10101}010110111010\ldots$, and we update $j = j'$.

If none of the conditions in Step 5 are fulfilled, we repeat Step 4, but since $j = 8$, we start looking, from the 8th index onwards, for an element of the set of L. We now find l_2 starting at the 12th index ($a = 10101010110\mathbf{111}010\ldots$). Then, $j' = 15$,

$$\mathbf{p} := 0^8 1^9 0^{10} 1^{11} 0^{12} 1^{13} 0^{14}$$

and $a := \mathcal{P}(a) = 1010101\mathbf{0101010}010\ldots$

Since $k \leq 3$ we may only repeat Step 4 one more time before generating a new string a with the GRNG.

5. One-way functions

A function $f : A^* \to B^*$ is one-way if it can be computed by a polynomial-time algorithm, but any polynomial-time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with negligible probability. Several candidates have withstood decades of intense scrutiny, hence the existence of one-way functions is an *open question*. The existence of a one-way function implies the existence of minimize-key encryption schemes secure against adaptive chosen-ciphertext attacks, message authentication codes, digital signature schemes.

The family of cryptographically secure hashing functions SHA is an example of possible one-way functions. To express the unpredictability of the outcomes, these types of functions often rely on Shannon entropy. As a smaller entropy is a symptom of less randomness, these functions aim to minimise the loss of entropy of their inputs. This is often realized by ensuring a normal distribution of outputs, that is, Borel-1 normality.

According to classical probability theory, if we toss a fair coin m times, we will draw any string in B with the same probability. However, for c compressible bit-string, by (1) we have

$$\{|x| = n : K(x) \leq n - c\} \cdot 2^{-n} \leq (2^{n-c+1} - 1) \cdot 2^{-n}\} < 2^{1-c}$$

That is, not all strings are equally probable since for large n, very few strings of length n are compressible. Also, note that the more a string deviates from the Borel-k normality bound, the more compressible it is.

Let x, y be two bit-strings of length m, let $\mathbf{p} \in P$ and let $x' = \mathcal{P}(x)$, $y' = \mathcal{P}(y)$. That is, the function $\mathcal{P}$ applies the pattern $\mathbf{p}$, of length r, to x and y by replacing only r bits. If x is more "random" than y, then x is less compressible than y. Moreover, x' and y' have the same bits replaced by the pattern $\mathbf{p}$, but the rest of the original bits were left untouched. For example, if $\mathbf{p} = 1^2 0^4 0^5$ then

$$x' = x_1 \mathbf{1} x_3 \mathbf{00} x_6 x_7 \ldots x_m \text{ and } y' = y_1 \mathbf{1} y_3 \mathbf{00} y_6 y_7 \ldots y_m.$$

Thus, with high probability, x' is less compressible than y'. As a consequence, x' is more likely to appear in the set of all strings of length m. In particular, it is more likely to appear in the set of all c-compressible strings of length m than y'.

Unfortunately, since $K(x)$ is incomputable [5], we cannot search through the set of c-compressible strings, *even this set is very small*. However, we can approximate the degree of randomness of a string by looking at computable symptoms of randomness properties such as Borel normality. In this context, we refer to strings with a similar degree of randomness as being in the same *band of randomness*. For example, all Borel-2 normal strings that are not Borel-3 normal, are in a different *band of randomness* than those that are Borel-3 normal.

For our task, we *do not know* what band of randomness the strings in T belong to. Nonetheless, for a randomized function such as those belonging to the SHA family, we *know* that given a highly random string as input, the output will be less random since some entropy will unavoidably be lost because we admit inputs larger than the output length, and cryptographic hash functions that are not surjective cannot maintain their input entropy. In addition, collision resistance only states that it is hard to find a collision, not that one does not exist.[2] However, this type of function aims to minimise entropy loss, allowing the use of the results above in the Randomised search, as illustrated

Consider the example at the end of Section 4.. Suppose that after Step 1, we see that with high probability a string in T is close to be Borel-2 normal. If f is entropy preserving, we only need to try input strings in the same band of randomness, i.e. strings close to Borel-2 normality. However, as discussed above, it is very likely that f will lose some entropy. So, we start with a

[2] `https://doi.org/10.6028/NIST.SP.800-175Br1.`

Borel-2 normal string and increasingly deviate from its normality bound to find the desired *band of randomness*. For instance, if $a = 10001010110111010\ldots$ is Borel-2 normal, then the first iteration increases the number of occurrences of “10” in the above prefix from 3 to 4, and by the second iteration, we get the string $a = 10101010101010010\ldots$, so “10” appears 7 times. Recalling that a string is Borel normal if all of its prefixes are Borel normal, we can see that this operation makes the input a deviate Borel-2 normality by a small margin; thus $f(a)$ is more likely to be in the same *band of randomness* as the strings in T. In contrast, we would need to try many more strings generated in lexicographic order to achieve the same result. If f is a function in the SHA family, then the first strings in lexicographic ordering have a lower entropy than the outputs since the function has a normal distribution of outputs. For example, $a = 000\ldots010$ is far from being Borel-2 normal or even Borel-1 normal, but $f(a)$ would likely be at least Borel-1 normal.

If we enumerate strings in lexicographic order, strings with high entropy will appear sparsely. However, GRNGs generate provably unpredictable strings, so by using them as inputs, we obtain highly random samples with high entropy. Note that PRNGs, which are computable, cannot guarantee the unpredictability of their generated strings, increasing the likelihood of having an output that is more random than the input. Consequently, if $f : A \to B$ minimises the loss of entropy and x, y are two strings of length n such that x is more random than y, then we expect $f(x)$ to be more likely to appear than $f(y)$ in the set of all strings of length m. Note that given highly random strings as inputs (such as the ones generated by a GRNG), we know that the outputs are in a *lower band of randomness*, due to the unavoidable loss of randomness applying f entails (potentially undetectable by statistical means).[3] Nonetheless, we do not know to what extent the quality of randomness is lost nor which *bands of randomness* the strings in T belong to.

6. An experiment

We use the Randomness search to explore the search space by consistently manipulating the degrees of randomness of the inputs and outputs through the controlled application of a relevant pattern. Each iteration reduces the randomness of the input in a controlled manner in order to search the target strings in a different *band of randomness*. If we go past the average Borel-k normality of the samples from T we are unlikely to be successful, so we stop and sample a new string. In this way we reduce the search space by sampling strings representative of a certain amount of randomness, controlled through the introduction of patterns and monitored via Borel normality.

To demonstrate we conducted the experiment described below using the function SHA-512.

Consider the following patterns:

$\mathbf{p_1} = 1^{150}1^{151}1^{152}1^{153}0^{154}0^{155}0^{156}0^{157}1^{158}1^{159}1^{160}1^{161}1^{162}0^{163}0^{164}0^{165}0^{166}1^{167}1^{168}1^{169}$,
$\mathbf{p_2} = 1^{55}1^{56}0^{57}0^{58}0^{59}0^{60}0^{61}0^{62}0^{63}0^{64}1^{65}0^{66}1^{67}0^{68}1^{69}0^{70}1^{71}0^{72}0^{73}0^{74}0^{75}$,
$\mathbf{p_3} = 0^{120}0^{121}0^{122}1^{123}1^{124}0^{125}0^{126}1^{127}1^{128}0^{129}0^{130}1^{131}1^{132}0^{133}0^{134}0^{135}1^{136}1^{137}0^{138}0^{139}$
$1^{140}1^{141}$,

[3] This is not always the case: consider a highly compressible string as input, such as the string of all 1’s and look at the output after going through a hashing function.

$\mathbf{p_4} = 1^{20}0^{21}1^{22}0^{23}0^{85}1^{86}1^{87}1^{88}0^{89}0^{400}0^{401}0^{402}1^{403}1^{404}$,
$\mathbf{p_5} = 0^{20}0^{21}1^{22}0^{85}1^{86}0^{86}1^{88}0^{400}0^{401}1^{402}0^{403}1^{404}$,
$\mathbf{p_6} = 1^{320}1^{321}1^{322}0^{340}0^{350}1^{351}0^{352}1^{353}0^{354}$.

Let $\mathbf{P_1} = \{\mathbf{p_1}\}$, $\mathbf{P_2} = \{\mathbf{p_2}, \mathbf{p_3}\}$, $\mathbf{P_3} = \{\mathbf{p_4}\}$, $\mathbf{P_4} = \{\mathbf{p_5}, \mathbf{p_6}\}$. Let $T_i \subset \{0,1\}^{512}$ be such that that every string in T_i has at least one pattern in $\mathbf{P_i}$. For each $\mathbf{P_i}$ we want to find the largest number of bit-strings a with $|a| = 512$, such that SHA-512$(a) \in T_i$, without exceeding $5,000,000$ calls to SHA-512 (acting as the time limit). We call a string a satisfying these requirements a *valid* string.

We compare the performance of conducting a Randomised search in finding valid strings against trying strings in lexicographic order. The metric is the number of unique valid strings that each method is able to find within the time limit. In this experiment, we apply a minor modification to the Randomised search: *instead of returning the first valid string found, we add it to the count and continue until the time limit has been reached.* Finally, we return the total amount of valid strings found. In the unlikely event that the same valid string appears again in a later iteration, we ignore it and continue the search.

Table 1. Experimental results for finding bit-strings a with $|a| = 512$, such that SHA-512$(a) \in T_i$. The last two columns represent the number of *valid strings* by each method.

Pattern set	Binary representation of	Lexicographic order	Randomised search
	lengths of patterns (L)		
T_1	$L = \{10100\}$	4	10
T_2	$L = \{10101, 10110\}$	3	6
T_3	$L = \{1110\}$	314	361
T_4	$L = \{1100, 1001\}$	10,652	11,236

Table 1 shows that the Randomised search is more efficient than the lexicographic order search because it can find more valid strings within the same number of calls to SHA-512.

Note that shorter patterns appear more frequently than longer ones: for example, for a string starting with a 1, it is natural to expect more valid strings if the patterns we look for are short. Consequently, we see a smaller amount of valid strings found for T_1 and T_2 compared to T_3 and T_4, but in all instances, we find more valid strings via Randomised search. In applications where each valid string found is noteworthy, this is a non-trivial advantage representing a significant reduction of the search space.

References

[1] M. Agrawal, N. Kayal, and N. Saxena, "Primes is in P," *Annals of Mathematics*, vol. 160, pp. 781–793, 2004.

[2] R. Solovay and V. Strassen, "A fast Monte-Carlo test for primality," *SIAM Journal on Computing*, vol. 6, no. 1, pp. 84–85, 1977. [Online]. Available: dx.doi.org/10.1137/0206006

[3] ——, "Erratum: A fast Monte-Carlo test for primality," *SIAM Journal on Computing*, vol. 7, no. 1, p. 118, 1978. [Online]. Available: dx.doi.org/10.1137/0207009

[4] C. Calude, *Information and Randomness—An Algorithmic Perspective*. Berlin: Springer, 2002 (2nd ed.).

[5] C. S. Calude, *To Halt or Not to Halt? That Is the Question*. World Scientific Publishing Co., Inc., Singapore, 2024.

[6] A. Kulikov, M. Jerger, A. Potočnik, A. Wallraff, and A. Fedorov, "Realization of a quantum random generator certified with the Kochen-Specker theorem," *Phys. Rev. Lett.*, vol. 119, p. 240501, Dec 2017. [Online]. Available: link.aps.org/doi/10.1103/PhysRevLett.119.240501

[7] J. M. Agüero Trejo and C. S. Calude, "A new quantum random number generator certified by value indefiniteness," *Theoretical Computer Science*, vol. 862, pp. 3–13, Mar. 2021. [Online]. Available: linkinghub.elsevier.com/retrieve/pii/S0304397520304679

[8] É. Borel, "Les probabilités dénombrables et leurs applications arithmétiques," *Rendiconti del Circolo Matematico di Palermo (1884 - 1940)*, vol. 27, pp. 247–271, 1909. [Online]. Available: dx.doi.org/10.1007/BF03019651

[9] C. Calude, "Borel normality and algorithmic randomness," in *Developments in Language Theory*, G. Rozenberg and A. Salomaa, Eds. Singapore: World Scientific, 1994, pp. 113–129.

[10] C. S. Calude, M. J. Dinneen, M. Dumitrescu, and K. Svozil, "Experimental evidence of quantum randomness incomputability," *Phys. Rev. A*, vol. 82, no. 2, p. 022102, Aug 2010. [Online]. Available: dx.doi.org/10.1103/PhysRevA.82.022102

[11] A. C. Martínez, A. Solís, R. D. H. Rojas, A. B. U'Ren, J. G. Hirsch, and I. P. Castillo, "Testing randomness in quantum mechanics," *CoRR*, vol. abs/1810.08718, 2018. [Online]. Available: arxiv.org/abs/1810.08718

[12] A. A. Abbott, C. S. Calude, M. J. Dinneen, and N. Huang, "Experimentally probing the algorithmic randomness and incomputability of quantum randomness," *Physica Scripta*, vol. 94, p. 045103, 2019.

[13] J. M. Agüero Trejo and C. S. Calude, "Photonic ternary quantum random number generators," *Proc. R. Soc. A*, vol. 479, pp. 1–16, 2023. [Online]. Available: doi.org/10.1098/rspa.2022.0543

[14] E. Barker, "Guideline for using cryptographic standards in the federal government: Cryptographic mechanisms," NIST Special Publication 800-175B Revision 1, Tech. Rep., Mach, 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-175Br1