PRIME CRAWLER: PSEUDO-RANDOM NUMBER GENERATOR

IVAN G. ZAIGRALIN

1. Summary

This section is intended for the general audience.

The Prime Crawler (or PC) is a pseudo-random number generator. It generates an infinite string consisting of zeroes and ones. The internal state of the generator may be represented by what we call a *pad*, which may be depicted as follows:

$$\begin{array}{l}\underline{0}01\\ \underline{0}1100\\ \underline{1}001100\end{array} \tag{1}$$

The pad consists of several so-called *lines*, which are fixed finite binary sequences (later treated as infinite periodic sequences). Notice that one number in each line is underlined, indicating the current position in that line. To generate a pseudo-random digit, PC counts the number of ones at underlined positions. If the number of ones is even, it generates 0, and if the number of ones is odd, it generates 1. In the state depicted above, it will generate 1. Readers familiar with the basic programming will recognize this a **xor**ing of the underlined digits. After generating a digit, PC shifts underlined positions to the right. If the end of a line is reached, it shifts the underlined position to the beginning of that line. Here are a few consecutive steps for the depicted pad:

$$\begin{array}{lc}
\text{State} & \text{Output}\\ \hline
\begin{array}{l}\underline{0}01\\ \underline{0}1100\\ \underline{1}001100\end{array} & 1\\ \hline
\begin{array}{l}0\underline{0}1\\ 0\underline{1}100\\ 1\underline{0}01100\end{array} & 1\\ \hline
\begin{array}{l}00\underline{1}\\ 01\underline{1}00\\ 10\underline{0}1100\end{array} & 0\\ \hline
\begin{array}{l}\underline{0}01\\ 011\underline{0}0\\ 100\underline{1}100\end{array} & 1
\end{array} \tag{2}$$

So the sequence generated by the depicted pad starts with 1101 and continues (potentially) forever. To exclude truly trivial cases, we demand that each line must contain at least one 0 and at least one 1. Additionally, we require that the line lengths

Date: 2014-01-22.

are pairwise distinct, relatively prime numbers. In the example given, the lengths are $3, 5, 7$. It comes out that just on these premises, the period of the generated sequence is the product of line lengths (in this specific case, the period is $3 \cdot 5 \cdot 7 = 105$), the period being the shortest repeating substring. For any PC, the sequence starts repeating itself as soon as the position markers find themselves in the initial state (1).

While using a pad (rather than a formula) in order to generate a pseudo-random sequence may seem inelegant, one has to recognize that even very small pads will result in very large periods. For example, for a pad on the first 100 primes, the period has 220 decimal places. It is easy to see that if n is the number of lines, the period grows faster than a^n, and from the Prime Number Theorem [7] one can see that it grows faster than $n!$.

A large period is, of course, irrelevant if the resulting sequence lacks "randomness" or "variability". It would be very fortunate if we could show that for some integer k, all possible binary strings of length k are approximately equidistributed, i.e. each of the 2^k different strings occurs about the same number of times within one period of the pseudo-random sequence. It would be even better if we could show that k could be made arbitrarily large by increasing n, the number of lines in the pad, and that we can get to high enough values of k without slowing down the computation too much or running out of the computer memory.

Another nice property we want to secure is being unpredictable. Ideally, one should not be able to predict the next digit with accuracy above 0.5, no matter how many outputs in a row one observes.

As it stands, we can show that binary digits are approximately equidistributed, and we have a good intuition about other very short strings. With a slight modification to the algorithm, we should be able to say the same about strings of length comparable to that of the longest line in a pad.

2. Notation

$\mathbb{N} = \{0, 1, 2, \ldots\}$.

Unless otherwise stated, *sequences* are indexed by $\mathbb{N}$.

A *string* is a finite sequence. A *substring* of a sequence is an improper subsequence which is also an interval with respect to the order topology. A *character* is synonymous with a *member of a sequence*.

3. Prime Crawler

A *pad* consists of n *lines* L_i, each line a binary sequence with the period p_i, $i = 0, 1, 2, \ldots, n-1$, with $p_i \neq 1$ pairwise distinct and relatively prime. Define $L_i(j)$ to be the j-th character in L_i, where $i = 0, 1, \ldots, n-1$ and $j \in \mathbb{N} = \{0, 1, 2, \ldots\}$. To generate a pseudo-random binary sequence $\langle r_j \rangle$, let

$$r_j = \bigoplus_{i=0}^{n-1} L_i(j),$$

where $\oplus$ is **xor**, or the *exclusive or*, i.e. $a \oplus b = 0$ if $a = b$, $a \oplus b = 1$ otherwise.

3.1. Period.

A constant line is not interesting, as a line of zeroes has no effect on r_j, a line of ones flips each r_j. We will assume that no line is constant (actually, it was already implicit in how we wanted periods to be different from 1). On that assumption, the period of $\langle r_j \rangle$, given some pad, is guaranteed to be

$$\prod_{i=0}^{n-1} p_i, \tag{3}$$

as will be shown shortly. In general, any such sequence $\langle r_j \rangle$ with period (3) will be called *the pad product of lines* L_i, $i = 0, 1, \ldots, n-1$ and will be written simply as a product $L_0 L_1 \ldots L_{n-1}$. It should be clear that the pad product is commutative and associative, as is $\oplus$.

The period is clearly bounded from above by (3); does it have to be that large? Yes. Fix L_0, L_1 with periods p_0, p_1 respectively. It will suffice to show that the period of $L_0 L_1$ is $p_0 p_1$ for p_0 and p_1 relatively prime and the induction will take care of the rest. Consider the initial segment of L_0 which is a concatenation of p_0 strings, each of length p_1. These strings cannot all be the same, as it would force L_0 to be constant. Hence the initial segment of $L_0 L_1$ of length $p_0 p_1$ cannot be a concatenation of p_0 identical strings. Similarly, the same initial segment cannot be a concatenation of p_1 identical strings. This establishes (3).

3.2. **Distribution of Digits.** We will say that the proportion of zeroes in a line is the proportion of zeroes in a periodic substring in that line.

Let $a_0, a_1, \ldots$ be proportions of zeroes in the respective lines. On our assumptions, $a_i \neq 0$ or 1. Also, $a_i = 1/2$ makes digits perfectly equidistributed in any pad product involving the respective line, so we will omit this case as trivial.

Let $b_i = a_i - 1/2$, so that $b_i \in (-1/2, 1/2)$, $b_i \neq 0$. Let c_i be the proportion of zeroes in the pad product of lines $L_0, \ldots, L_i$. Let $d_i = c_i - 1/2$. In particular, $c_0 = a_0$ and $d_0 = b_0$.

In general, given two lines with proportions of zeroes a_0 and a_1, the proportion of zeroes in their pad product is

$$a_0 a_1 + (1 - a_0)(1 - a_1) = 2a_0 a_1 + 1 - a_0 - a_1.$$

In our case, the i-th pad product can be obtained from the $(i-1)$-st pad product and the i-th line, so for $i > 0$ we have

$$\begin{aligned}
c_i &= 2c_{i-1}a_i + 1 - c_{i-1} - a_i, && (4)\\
d_i + 1/2 &= 2(d_{i-1} + 1/2)(b_i + 1/2) + 1 - (d_{i-1} + 1/2) - (b_i + 1/2) && (5)\\
d_i &= 2d_{i-1}b_i = 2^i \prod_{j=0}^{i} b_j. && (6)
\end{aligned}$$

Since $b_i \in (-1/2, 1/2)$ for all $i \in \mathbb{N}$, $|d_i|$ is strictly decreasing for every pad. Note that $d_i \neq 0$ for all $i \in \mathbb{N}$.

3.2.1. *Worst Case Scenario.* If digits are to be approximately equidistributed, c_i has to approach 1/2 and d_i has to approach 0. The "worst case" clearly happens when each $|b_i|$ is as large as possible, i.e. when each line contains but one 0 or but one 1. The maximum period length among first i lines of the pad grows at least as fast as primes, and possibly faster. It is easy to come up with an example where period lengths grow too fast for d_i to converge to zero, so let us suppose that period lengths are primes above 2.[1]

If we agree to write ρ_i for i-th prime, then the worst case could be conceived as

$$b_i = \frac{1}{2} - \frac{1}{\rho_{i+2}},$$

and we need to inspect what happens to d_i as $i \to \infty$. (We wrote ρ_{i+2} because i counts from zero while the first prime useful to us is $\rho_2 = 3$.) Well, in agreement with (6), d_i converges to zero as $i \to \infty$ iff the following product converges to zero:

$$\prod_{i=1}^{\infty} 2\left(\frac{1}{2} - \frac{1}{\rho_{i+1}}\right) = \prod_{i=1}^{\infty}\left(1 - \frac{2}{\rho_{i+1}}\right). \tag{7}$$

What we have in (7) is the multiplicative inverse of

$$\prod_{i=1}^{\infty} \frac{1}{1 - 2\rho_{i+1}^{-1}}, \tag{8}$$

with the latter diverging to infinity just in case if the former approaches zero. Which it does, and we prove it by showing that a slower-growing product diverges:

$$\prod_{i=1}^{\infty} \frac{1}{1-\rho_i^{-1}} = \left(1 + \tfrac{1}{2} + \tfrac{1}{2^2} + \ldots\right)\left(1 + \tfrac{1}{3} + \tfrac{1}{3^2} + \ldots\right)\left(1 + \tfrac{1}{5} + \tfrac{1}{5^2} + \ldots\right)\ldots \tag{9}$$

$$= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots \tag{10}$$

$$= \infty. \tag{11}$$

We may conclude that in the case when line periods grow as slow as primes, the proportion of zeroes in a pad product approaches 0.5 as the number of lines tends to ∞.

3.2.2. *A More Reasonable Case.* Suppose that we have a somewhat more "typical" pad, one in which the proportion of zeroes in each line is at least 1/4, and hence $b_i \leq 1/4$. This assumption is true of most lines in most pads. Then

$$|d_i| = \left|2^i \prod_{j=0}^{i} b_j\right| \tag{12}$$

$$\leq 2^i \prod_{j=0}^{i} \frac{1}{4} \tag{13}$$

$$= \frac{1}{2^{i+2}}, \tag{14}$$

i.e. in a "typical" pad, where zeroes and ones are approximately equidistributed, the proportion of zeroes in the pad product approaches $\frac{1}{2}$ very fast. In practice, one can easily compute an upper bound on $|d_i|$ in a given pad by counting lines with proportion of zeroes at least 1/4.

[1] Which is the *best case* scenario with respect to the period growth, but it is also the most practical one if we actually *want* to have as many lines as possible, given a fixed amount of computer memory.

3.3. **Distribution Of Strings.** We start by introducing an auxiliary concept of a *blurring random process* and proving things about it. We conclude by proving the main theorem in this section, which characterizes distribution of strings of a given length in the period of our number generator.

3.3.1. *Blurring Walks.* We will define a special kind of a random walk on a set of two elements: at time $i \in \mathbb{N}^+$, the traveler will either go to the other element in the set or will remain where it is. Specifically, let $T_0, T_1, T_2, T_3, \dots$ be a sequence of random variables with values among $X = \{0, 1\}$ and let $T_{i+1} = 1 - T_i$ with probability p_i and $T_{i+1} = T_i$ with probability $1-p_i$. Let $T_0 = 0$. Of primary interest to us is the distribution of T_i as $i \to \infty$. We can write the distribution of T_0 as a row vector

$$D_0 = (1, 0),$$

indicating that $P(T_0 = 0) = 1$ and $P(T_0 = 1) = 0$, and then compute D_i by recursion:

$$D_i = D_{i-1} \begin{pmatrix} 1-p_i & p_i \\ p_i & 1-p_i \end{pmatrix}.$$

Definition 3.3.2. Let the process $T_0, T_1, T_2, \dots$ as defined above be called a *blurring process* and the corresponding sequence $p_1, p_2, p_3, \dots$ a *blurring sequence* iff

$$\lim_{i\to\infty} D_i = (1/2, 1/2).$$

Of course, not every sequence is a blurring sequence: for example, $p_i \equiv 0$ will result in $\lim_{i\to\infty} D_i = (1, 0)$. In general, however, non-trivial values p_i will drag the distribution in the right direction, but not always fast enough. To show what we mean, write

$$D_i = D_0 \begin{pmatrix} 1-q_i & q_i \\ q_i & 1-q_i \end{pmatrix} = D_0 \prod_{k=1}^{i} \begin{pmatrix} 1-p_k & p_k \\ p_k & 1-p_k \end{pmatrix}.$$

Here we can see that $q_1 = p_1$ and furthermore, to obtain the next q we calculate

$$\begin{pmatrix} 1-q_i & q_i \\ q_i & 1-q_i \end{pmatrix} \begin{pmatrix} 1-p_{i+1} & p_{i+1} \\ p_{i+1} & 1-p_{i+1} \end{pmatrix} = \begin{pmatrix} 1-(q_i+p_{i+1}-2q_ip_{i+1}) & (q_i+p_{i+1}-2q_ip_{i+1}) \\ \cdots & \cdots \end{pmatrix},$$

which means that $q_{i+1} = q_i + p_{i+1} - 2q_ip_{i+1}$. The reader may notice similarities with the work we did for distribution of digits. Indeed, it is the same algebra, but we need the new language and we will take it a bit farther.

This last relation for q_i gives us a sense of what is happening: we hope that each successive p_{i+1} tugs q_i towards $1/2$, and now we can check it and compute by how much.

$$\begin{aligned} \frac{1/2 - q_{i+1}}{1/2 - q_i} &= \frac{1/2 - (q_i + p_{i+1} - 2q_ip_{i+1})}{1/2 - q_i} \\ &= 1 - 2p_{i+1}. \end{aligned}$$

Since $p_k \in [0, 1]$ for all $k \in \mathbb{N}^+$, the "shrinking factor" for the distance from q_i to $1/2$ is

$$|1 - 2p_{i+1}| \le 1.$$

Moreover,

$$1/2 - q_i \to 0 \text{ if } \prod_{k=1}^{i} |1 - 2p_i| \to 0. \tag{15}$$

Notice that the sign under the absolute value is somewhat irrelevant. What makes a sequence blurring is how close it remains to $1/2$. In particular, *Fact* 3.3.3. If the sequence $p_1, p_2, p_3, \dots$ is a blurring sequence, and a sequence r_i has the property that $|1/2 - r_i| \le |1/2 - p_i|$, then r_i is a blurring sequence. One can think of r_i as "bounded in the neighborhood of 1/2" by p_i. An easy consequence is that any sequence $r_1, r_2, r_3, \dots$ where $r_i = p_i$ or $1 - p_i$ is a blurring sequence. All of this is immediate from (15).

We are now ready to treat that very particular and useful to us sequence.

Claim 3.3.4. *Let ρ_i be the i-th prime, $i \in \mathbb{N}^+$ and let k be a positive integer. Then the sequence $p_i = k^{-1}\rho_i^{-1}$ is a blurring sequence.*

Proof. In view of (15), it suffices to show that

$$\prod_{i=1}^{\infty}(1 - 2k^{-1}\rho_i^{-1}) = 0,$$

or, equivalently, that

$$\prod_{i=1}^{\infty}\frac{1}{1-2k^{-1}\rho_i^{-1}} \ge \prod_{i=1}^{\infty}\frac{1}{1-k^{-1}\rho_i^{-1}} = \infty.$$

In the spirit of (9), we can write

$$\begin{aligned}\prod_{i=1}^{\infty}\frac{1}{1-k^{-1}\rho_i^{-1}} &= \left(1+\tfrac{1}{2k}+\tfrac{1}{2^2k^2}+\dots\right)\left(1+\tfrac{1}{3k}+\tfrac{1}{3^2k^2}+\dots\right)\dots \\ &= 1+\tfrac{1}{k}\left(\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{5}+\dots+\tfrac{1}{\rho_i}+\dots\right)+\dots \\ &= \infty,\end{aligned}$$

since the series $\sum_{i=1}^{\infty}\frac{1}{\rho_i}$ diverges ([1, 17]). For a classical treatment see ([1, 349-351]. □

With the claim secured, we are ready to introduce the main result in this section.

Theorem 3.4. *Let L_i be an arbitrary sequence of pad lines, $i \in \mathbb{N}$, period of L_i is ρ_i, the i-th prime. Let n be an arbitrary positive integer. Then the distribution of strings of length n in the period of the pad on the first i lines approaches uniform distribution as $i \to \infty$.*

Proof. To outline the proof, we will represent the probability space associated with the pad product by that of a certain random walk among the vertices of the n-dimensional cube. We will then project this walk onto many different partitions of the cube to obtain induced random walks on $\{0,1\}$. All these walks will be blurring, and so we will gather all of their resulting relations and make a conclusion about the overall distribution.

Look at a binary string of length n as a vertex of the n-dimensional cube. Our random walk will start at the origin, time will start at zero. If our coordinates are $\mathbf{x}_{i-1}$ at time $i-1$, then we choose fairly one of the ρ_i substrings of length n out of L_i, call it $\mathbf{d}_i$, and go to the vertex $\mathbf{x}_i = \mathbf{x}_{i-1} \oplus \mathbf{d}_i$. For any fixed i and a walk up to time i, the resulting probability space is identical to that of the pad on the first i lines. Let W_i be the position in the cube at time i; this is the walk we will be projecting.

Now we will show how to divide the cube into two parts, in $2^n - 1$ different ways. Actually, look at the figure 3.3.1 first, it shows the partitions for $n = 3$. (For the sake of notational clarity, we will continue to abuse the case when $n = 3$ until the end of this proof, but everything we say will extend naturally to the general case.)

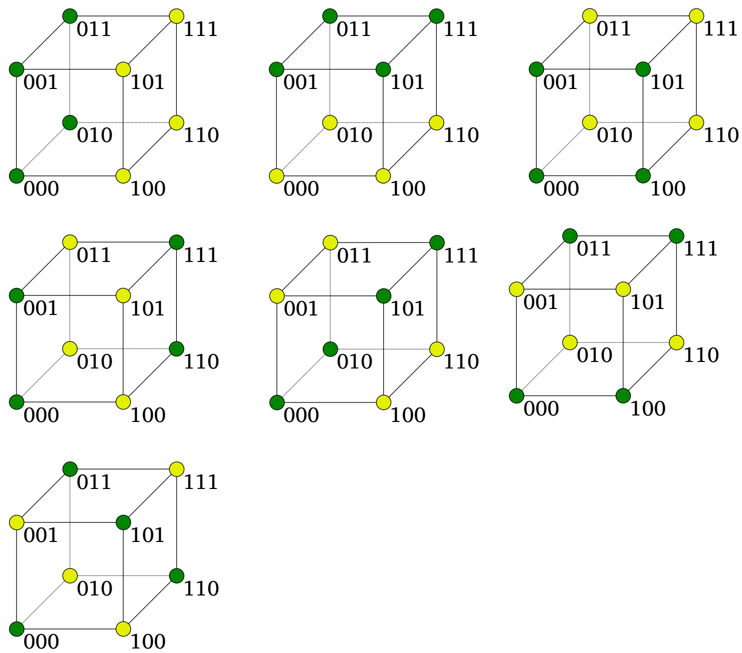

FIGURE 1. Partitions of the 3-dimensional cube.

Partitions in the first row of the figure are obtained by fixing the parity of a single coordinate; the ones in the second row by fixing the sum $(\mathrm{mod}\, 2)$ of two coordinates (e.g. setting the sum of the first two coordinates to zero yields half of the cube: $000, 001, 110, 111$); the one in the last row by fixing the parity of the sum of all three coordinates. The quantities in each row are clearly binomial coefficients, which is why we have exactly $2^n - 1$ partitions.

Next, we project our random walk onto each one of these partitions. Let T_i, $i \in \mathbb{N}$, be one of these projections. The range of T_i can be thought of as $\{0, 1\}$, where 0 corresponds to the "even" half of the cube, and 1 corresponds to the "odd" half. We claim that T_i is a blurring process. To make sure, let us fix i and inspect the probability p_i of T_i going to a different half of the cube. We will show that for all $i \geq 2^{n-1}$, this probability p_i is at least ρ_i^{-1} and at most $1 - \rho_i^{-1}$, and then T_i will be blurring by Claim (3.3.4). Certainly, T_i can send us to at least two and at most ρ_i different places, so we only need to show that not all of those places are in the same half of the cube. The instructions for where to go are given in a string of length n, which is a substring of L_i (it is OK to wrap around), write it as $(\delta_1, \delta_2, \ldots, \delta_n)$. Our partition, on the other hand, is determined by the parity of certain coordinates, so let us represent it as a vector $(l_1, l_2, \ldots, l_n)$, where $l_\xi = 1$ iff ξ is one of the coordinates that matter. Then

$$z = l_1\delta_1 + l_2\delta_2 + \ldots + l_n\delta_n \ (\mathrm{mod}\, 2)$$

tells us exactly what we want: $z = 1$ iff $\boldsymbol{\delta}$ sends us to a different half of our partition. For contradiction, suppose that z is constant when computed for each of the ρ_i

substrings of L_i. Let δ_ξ now be a digit of L_i, so that we can write

$$l_1\delta_k + l_2\delta_{k+1} + \ldots + l_n\delta_{k+n-1} = l_1\delta_{k+1} + l_2\delta_{k+2} + \ldots + l_n\delta_{k+n} \pmod 2, \tag{16}$$

where k is any natural number. Without loss of generality, $l_n = 1$, so what we have here is a linear recurrence relation for δ_{k+n} with period ρ_i. But the largest period this linear recurrent relation can have is $2^{n-1} - 1$, so we have a contradiction and have to conclude that z cannot be the same on the entire line, and hence T_i is a blurring process.

For the last leg of this proof, let $x_1^i, x_2^i, \ldots, x_{2^n}^i$ be the probabilities for being in a certain vertex of the cube after a random walk over the vertices of the cube at time i, i.e. values from the pmf of W_i. For each of the $2^n - 1$ ways to partition the cube we can write a statement saying that a corresponding projection is blurring. In case when $n = 3$, we get

$$\begin{array}{rcl} x_1^i + x_2^i + x_3^i + x_4^i - x_5^i - x_6^i - x_7^i - x_8^i & \to & 0, \\ x_1^i + x_2^i - x_3^i - x_4^i + x_5^i + x_6^i - x_7^i - x_8^i & \to & 0, \\ & \vdots & \\ x_1^i - x_2^i + x_3^i - x_4^i + x_5^i - x_6^i + x_7^i - x_8^i & \to & 0, \end{array}$$

where the arrow is for $i \to \infty$. And we get this one for free:

$$x_1^i + x_2^i + x_3^i + x_4^i + x_5^i + x_6^i + x_7^i + x_8^i = 1.$$

The vector $\mathbf{x}^i$ is bounded, so it has a limit point $\mathbf{x}$, which must be the unique (!) solution to the system of equations, as above, with equalities instead of arrows. As an illustration ($n = 3$), sum up everything to get

$$2^n x_1 = 1.$$

Other components are obtained similarly, and it works just as well for any $n \in \mathbb{N}^+$. We have shown that the pmf of W_i tends to be uniform as $i \to \infty$, and so does the corresponding distribution of strings of length n in the pad product on first i lines. $\square$

Note 3.4.1. In his 1965 paper ([3]), Robert C. Tausworthe describes a pseudo-random number generator $\langle a_k \rangle$, a $(0, 1)$ sequence generated by an n-th degree maximal-length linear recurrence relation modulo two. The period in his case was $2^n - 1$, just as for (16) in the proof above.

4. Further Directions

There are two major directions of further inquiry we can mention.

One is the exploration of the Prime Crawler's cryptographic fitness. On the face of it, the PC is ill-suited for cryptographic purposes [4]. In particular, if the internal state (the pad) of the PC is exposed, then an attacker can run the generator both forward and backward with perfect ease. But questions still remain with respect to the PC's passing of the next-bit test [5]. Here, an attacker's goal is to predict the future outputs based on the previous ones. So fix a pad with n lines of lengths $p_0, p_1, \ldots, p_{n-1}$ respectively. A naive attacker can reconstruct the state by observing $k = p_0 + p_1 + \ldots + p_{n-1}$ consecutive outputs, call them $r_0, r_1, \ldots, r_{k-1}$. The lines can then be reconstructed by solving a system of k linear equations:

$$\begin{aligned}L_0(0)+L_1(0)+\ldots+L_{n-1}(0) &= r_0\\ L_0(1)+L_1(1)+\ldots+L_{n-1}(1) &= r_1\\ &\ldots\\ L_0(k-1(\operatorname{mod} p_0))+L_1(k-1(\operatorname{mod} p_1))+\ldots+L_{n-1}(k-1(\operatorname{mod} p_{n-1})) &= r_{k-1}\end{aligned}$$

Solving systems of linear equations can be done in polynomial (sub-cubic) time, but in our case of Boolean variables, more specialized algorithms exist [2].

Another, sexier, but quite possibly hopeless direction to explore is the relationship between the Prime Crawler and the function $\omega(n)(\operatorname{mod} 2)$, where $\omega(n)$ is the number of distinct prime factors of n. A "sparse" pad with consecutive primes for line lengths, and with a single 1 in the end of each line, will faithfully generate the initial segment of $\omega(n)(\operatorname{mod} 2)$, and an infinite pad will generate the entire sequence, so a finite pad may be though of as printing a "periodic approximation" of $\omega(n)(\operatorname{mod} 2)$. As the pad grows larger, the distribution of substrings becomes more fair, so there may be a way here to show that $\omega(n)(\operatorname{mod} 2)$ is a normal number in base 2 [6].

5. Acknowledgments

Thanks to Peter Barendse for a stimulating discussion related to the Euler Product [8], and to my former office mate Myoungil Kim for the slick algebraic tricks (9).

6. Copying

Copyright 2009, 2014 Ivan Zaigralin. This document is licensed under Attribution 4.0 International license, full text at http://creativecommons.org/licenses/by/4.0/. In short, you are free to share (copy and redistribute the material in any medium or format) and adapt (remix, transform, and build upon the material for any purpose, even commercially) this work, as long as you abide by the terms of the license.

References

[1] G.H. Hardy and E.M. Wright. *An Introduction To The Theory Of Numbers*. Oxford University Press, Ely House, London, fourth edition, 1960.

[2] A.E. Litvinenko. Solving systems of linear equations with boolean variables. *Cybernetics and Sys. Anal.*, 42(5):649–655, September 2006.

[3] Robert C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19(90):201–209, 1965.

[4] Wikipedia. Cryptographically secure pseudorandom number generator. https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator. [Online; accessed 2014-04-03].

[5] Wikipedia. Next-bit test. https://en.wikipedia.org/wiki/Next-bit_test. [Online; accessed 2014-04-03].

[6] Wikipedia. Normal number. https://en.wikipedia.org/wiki/Normal_number. [Online; accessed 2014-04-03].

[7] Wikipedia. Prime numbet theorem. https://en.wikipedia.org/wiki/Prime_number_theorem. [Online; accessed 2014-04-03].

[8] Wikipedia. Riemann zeta function. https://en.wikipedia.org/wiki/Riemann_zeta_function. [Online; accessed 2014-04-03].