J Inf Process Syst, Vol.13, No.6, pp.1554~1574, December 2017
https://doi.org/10.3745/JIPS.01.0021

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Generation of Finite Inductive, Pseudo Random, Binary Sequences

Paul Fisher*, Nawaf Aljohani**, and Jinsuk Baek*

Abstract

This paper introduces a new type of determining factor for Pseudo Random Strings (PRS). This classification depends upon a mathematical property called Finite Induction (FI). FI is similar to a Markov Model in that it presents a model of the sequence under consideration and determines the generating rules for this sequence. If these rules obey certain criteria, then we call the sequence generating these rules FI a PRS. We also consider the relationship of these kinds of PRS's to Good/deBruijn graphs and Linear Feedback Shift Registers (LFSR). We show that binary sequences from these special graphs have the FI property. We also show how such FI PRS's can be generated without consideration of the Hamiltonian cycles of the Good/deBruijn graphs. The FI PRS's also have maximum Shannon entropy, while sequences from LFSR's do not, nor are such sequences FI random.

Keywords

Pseudo Random, Linear Shift Registers, Finite Induction, Graphs, Hamiltonian Cycles

1. Introduction

There is a quote from John von Neumann [1] in which he states that producing random digits from an arithmetical method would be equivalent to someone in a state of sin. Measuring the randomness of any single digit is impossible, and so we set about to measure the randomness of a sequence of numbers. This sequence should be infinite, and if random the amount of information carried in the sequence in terms of Shannon entropy [2] should be likewise infinite. However it is difficult to deal with infinite 'anythings', so random strings are finite, but perhaps very long, or with very long periods. Or such strings can be generated by processes that are themselves random. Such processes can come from physical processes, quantum processes, or mathematics. Examples of such physical generators of digits arise from interesting sources. For example, techniques built around quantization of interstellar noise, coin tossing of an idealized coin for binary sequences, quantum processes, radioactive decay of an element are such examples.

Since we need random numbers for many applications ranging from simulation to cryptology, we must have at our disposal random number generators that provide rapid results that may not be perfectly random, but whose results are close enough to random to be of value in the applications

Manuscript received May 19, 2015; accepted September 25, 2016.
Corresponding Author: Jinsuk Baek (baekj@wssu.ed)
* Dept. of Computer Science, Winston-Salem State University, Winston-Salem, NC, USA (fisherp@wssu.edu, baekj@wssu.edu)
** Institute of Public Administration, Riyadh, Saudi Arabia (nawaf9900@hotmail.com)

desired. These almost random sequences are called pseudo random sequences, and they arise from one of several classes of algorithms implemented in software or hardware. The most common class of algorithms for these pseudo sequences are generated by linear feedback systems [3], among several other techniques.

We introduce in this paper, a new method and test for both generating and examining random sequences. The method is based upon Hamiltonian cycles in directed graphs constructed using a particular technique. Since the length of a Hamiltonian cycle depends upon the number of vertices within a graph, theoretically one can construct a cycle whose period is as long as is desired. Further, the number of Hamiltonian cycles in a large graph can truly be astounding, the potential sequences generated by such a technique, as will be defined, provides a rich landscape for these kinds of random sequences.

The remainder of this paper is outlined as follows. Section 2 reviews the graph structure for generating the kind of sequences desired, and relates this work to linear feedback systems. Section 3 of this paper introduces the idea of finite inductive sequences. Section 4 deals with the generation of di-graphs based upon a construction technique to be described, providing several examples. Section 5 applies various tests designed to determine randomness of a sequence to those we have generated. We conclude in Section 6 with a summary of the work presented in this paper.

2. Review of Graph Issues and Their Relation to Linear Feedback Systems

The following section reviews some information relevant to understand the remainder of the paper.

2.1 Review of Graphs

In 1946 two papers were published describing a special kind of graph. The authors Good [4] and deBruijn [5] defined a type of directed graph which had certain properties of some curiosity. The definition of a deBruijn graph of k symbols is a directed graph with k^n vertices consisting of all possible length n sequences of the k symbols. In this graph, each vertex has k incoming and k outgoing edges. Each deBruijn graph is both Eulerian and Hamiltonian. It is the Hamiltonian cycles that are of interest.

In [6] we constructed deBruijn graphs by considering the binary representation of vertices in the graph of 2^m labels, where m is an integer number. The edges were defined by shift (S) and shift complement (SC) operations where the S was a circular left shift, and the SC was just the complement of the shifted bit. We also point out that the direction of the shift is immaterial, as a circular right shift and associated complement will produce the same results. Table 1 shows the values for 2^3 vertices and the edges that are generated by the shifting operations.

From Table 1, we can now construct the deBruijn digraph. For example in Table 1, vertex 000 connects to vertex 000 and 001. Fig. 1 shows the graph.

The graph of Fig. 1 has some other interesting properties. The radius of the graph is 3. The radius is defined [7] as follows. For a fixed vertex $v \in V$ and all other vertices $w \in V$, the integer $R(v)$ is defined to be the minimal distance to the most remote of vertices from v. It is given by the formula (1).

Table 1. A graph with eight vertices derived from the shifting operations

Vertex	Shift (S)	Shift complement (SC)
000	000	001
001	010	011
010	100	101
011	110	111
100	001	000
101	011	010
110	101	100
111	111	110

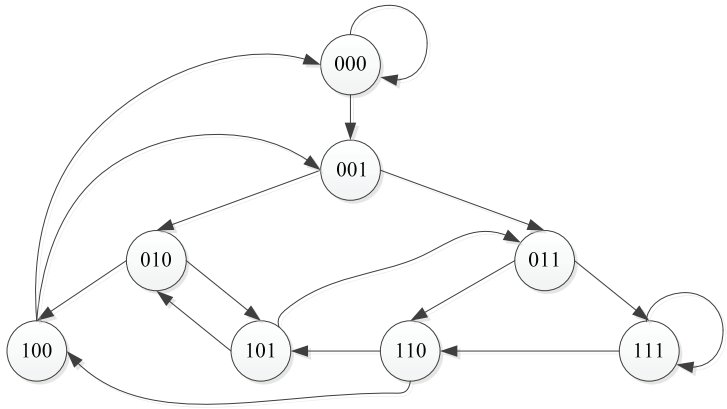

Fig. 1. deBruijn graph of eight vertices constructed from Table 1.

$$R(v) = \max_{w \in V} d(v, w) \tag{1}$$

As can be seen from Fig. 1, the radius is three or in the more general case of 2^n labels, the radius will be n. The deBruijn graph is also strongly connected as there is a path between every pair of vertices. The two distinct Hamiltonian cycles for the graph of Fig. 1 are given in (2a) and (2b). Two Hamiltonian cycles, or deBruijn sequences, of a graph are considered distinct if one cannot be generated from the other by applying a circular permutation [8].

$$000 \rightarrow 001 \rightarrow 010 \rightarrow 101 \rightarrow 011 \rightarrow 111 \rightarrow 110 \rightarrow 100 \rightarrow 000 \tag{2a}$$

$$000 \rightarrow 001 \rightarrow 011 \rightarrow 111 \rightarrow 110 \rightarrow 101 \rightarrow 010 \rightarrow 100 \rightarrow 000 \tag{2b}$$

From these two cycles, we see that there are a total of 16 such cycles generated from the two sequences of (2) and the cycles are obtained by starting at any vertex in the sequence and ending on that vertex. These additional cycles are simply circular shifts of the vertices in (2) with adjustments made to accommodate the starting/ending vertex. It is also interesting to note in the sequences of (2), that the bits in the identical positions in the vertices also have a periodicity. In (3a) and (3b) we have written the first, second and third bits respectively in sequence from each of the vertices in (2a) and then in (2b) respectively, excluding the ending vertex, as they appear.

1st bits: 0 0 0 1 0 1 1 1 2nd bits: 0 0 1 0 1 1 1 0 3rd bits: 0 1 0 1 1 1 0 0 (3a)

1st bits: 0 0 0 1 1 1 0 1 2nd bits: 0 0 1 1 1 0 1 0 3rd bits: 0 1 1 1 0 1 0 0 (3b)

In [9] the authors describe the recursive construction of a form of generalized, non-binary deBruijn sequence. They define a group Z_k of residues modulo k. This group is composed of the set of vectors k^m of length m. An order m deBruijn sequence with alphabet in Z_k is a sequence that includes all possible substrings of size m exactly once. Further they give a technique to construct these more general sequences when k is two or greater.

In [10] the authors define again the more general form of deBruijn graphs and provide some additional features of these graphs. Consider the graph G and integers $k > 1$ and $m \geq 1$, then G is an oriented graph (the more general deBruijn graph) of order m if for any ordered pair of vertices, there exists a unique directed path of length m from the first vertex to the second one. That is, the radius of the oriented graph must be m. The features for these oriented graphs are established by a theorem in [10] with the following conclusions:

a. The number of vertices of G is the m^{th} power of k.
b. Each vertex of G has out-degree k.
c. Each vertex of G has in-degree of k.
d. The number of loops in G is k.

A loop is defined to be present when any vertex connects to itself with no intervening vertices on the path. This is equivalent to the number of ones on the main diagonal (trace) of the adjacency matrix for the graph. As seen from Fig. 1, there are two such loops.

In [8] a count of the number of cycles is formulated for these oriented graphs that generate deBruijn sequences. This count is determined by the formula given in (4).

$$\text{Num} = A_i \prod_{j=1}^{n} (k_j - 1)! \ , \tag{4}$$

where k is the out degree of vertex j, and A_i is the number of spanning arborescences with root v_i where the v_i are vertices in the oriented graph. A spanning arborescence [11] on a directed graph starts with a root node v_i and constructs an elementary path (a path using all vertices in the graph exactly once) from the root node to every other node v_j where v_j is distinct from the root node v_i (a Hamiltonian path). For the oriented graph of Fig. 1, we see Num is 16, since the k_j are all 2, so the product is simply 1, and the arborescence for A_i is likewise 2 for each root vertex, and there are a total of 8 potential root vertices.

2.2 deBruijn Graphs and Linear Feedback Systems

If we consider the relation of deBruijn sequences to Linear Feedback Shift Registers (LFSR) [3], then for the oriented graph of Fig. 1, the primitive equation would then be $X^3 + X + 1$. This LFSR is shown in Fig. 2. The flip flops are clocked, and values shift from left to right each clock cycle.

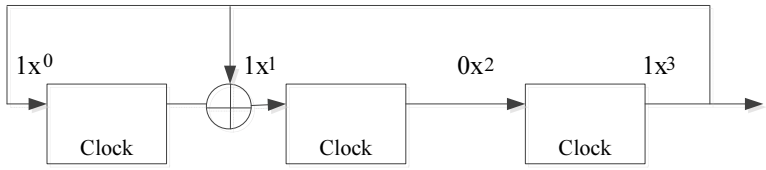

Fig. 2. D-Flip flop circuit for the graphs of Fig. 1.

If the three flip flops are initialized with all 1's as shown in Fig. 2, then the output of this circuit is 1 1 0 0 1 0 1, and the internal values of these three flip flops are given in (5):

$$1\,1\,1 \quad 1\,0\,1 \quad 1\,0\,0 \quad 0\,1\,0 \quad 0\,0\,1 \quad 1\,1\,0 \quad 0\,1\,1 \quad 1\,1\,1\ldots \tag{5}$$

We see that the output has four ones and three zeros, as the all zero state in Fig. 2 is not allowed, since once this state is defined, the circuit in Fig. 2 will get stuck in that state. The properties [12] of such LFSRs are as follows due to the exclusion of the zero state:

- Number of 1's equals the number of 0's + 1.
- 1/2 have runs of length 1.
- $1/2^2$ have runs of length 2.

  . . .
- $1/2^n$ have length n as long as $1/2^n$ as long as the number of runs indicated exceeds 1.
- The auto-correlation function $C(r)$ is a two valued function defined as follows where a_n is the symbol in the sequence at position n, and p is the number of symbols in the sequence,

$$C(r) = \sum_{n=1}^{p} a_n a_{n+r} = \begin{cases} \dfrac{p}{2} & \text{if } r = 0 \\ k < \dfrac{p}{2} & \text{if } 0 < r < p \end{cases}$$

The auto-correlation function must be maximum when $r = 0$.

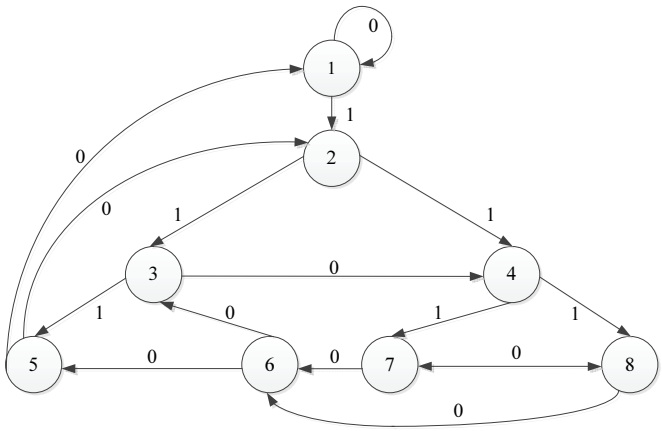

Fig. 3. Di-Graph of eight vertices with an in- and out-order of two.

3. Finite Inductive Sequences

Consider the digraph shown in Fig. 3 containing eight vertices and depth three and each vertex has in-degree and out-degree of two. There are four Hamiltonian cycles associated with the graph of Fig. 3 given in (6). Fig. 3 does not represent a deBruijn graph.

$$1\,2\,3\,4\,7\,8\,6\,5\,1 \quad 1\,2\,3\,4\,8\,7\,6\,5\,1 \quad 1\,2\,4\,7\,8\,6\,3\,5\,1 \quad 1\,2\,4\,8\,7\,6\,3\,5\,1 \tag{6}$$

We can represent these four sequences using a simple transformation by following the cycle and instead of recording the vertex number, record the label on the edge between the two vertices. We will call this sequence, derived from the Hamiltonian cycle, the label cycle. Providing this substitution, we obtain the label cycles as shown in (7). We note that the label cycles are similar to the output of the LFSR except here we allow the zero state, so the sequence length is 2^n not 2^n–1.

$$1\,1\,0\,1\,0\,0\,0\,0 \quad 1\,1\,0\,1\,0\,0\,0\,0 \quad 1\,1\,1\,0\,0\,0\,1\,0 \quad 1\,1\,1\,0\,0\,0\,1\,0 \tag{7}$$

We see that the label cycles are identical for the first and second pair of Hamiltonian cycles of (1). The objective is now to develop a technique to determine if the label cycles represent random sequences of a particular type which we will call finite inductive (FI) random sequences.

A FI sequence is any sequence, finite or infinite, which can be represented by a set of implicants having a reduced inductive base (IB). In order for an infinite sequence it must be eventually periodic [6]. This IB of a sequence is similar to the order of a Markov Model for that sequence. An implicant is the minimal precursor subsequence of adjacent symbols in a sequence that uniquely specifies the next symbol. The form of an implicant is:

$$\text{antecedent} \rightarrow \text{consequent},$$

where the antecedent is the minimal precursor subsequence, and the length of the antecedent is the IB for that implicant. The consequent is the next symbol following the right most symbol of the antecedent. The antecedent can be arbitrarily long (as long as the subsequence from the initial symbol to the rightmost symbol of the antecedent), but we will always assume that the antecedent is in a reduced form, that is any unnecessary initial symbols have been stripped off so only the necessary symbols are kept in the antecedent to make the antecedent unique for that consequent. Thus, the IB for the implicant is always as small as possible. The IB for the entire sequence is then the maximal IB (MIB) for the reduced set of implicants. The number of implicants will be bounded above by the length of the string being considered. Prime implicants are those initially obtained without constraint on the IB.

The process, which we call factoring, for obtaining these implicants is accomplished by an algorithm running in O(n) time where n is the length of the sequence to be processed. If there is one implicant whose IB is less than the MIB for these prime implicants, then it is possible to change the MIB to another value less than the MIB for the prime implicants [6].

To illustrate the concept of factoring, consider the first label sequence in (7). We treat the label sequence as one period and so the sequence to be processed is shown in (8) where the symbol '||' indicates the start of the next period. First we determine the prime implicants, and these are shown in (9).

$$1\,1\,0\,1\,0\,0\,0\,0 \,||\, 1\,1\,0\,1\,0\,0\,0\,0 \,||\, \ldots \tag{8}$$

$$\begin{array}{llll} 1\,1{\rightarrow}0 & 1\,1\,0{\rightarrow}1 & 1\,0\,1{\rightarrow}0 & 0\,1\,0{\rightarrow}0 \\ 1\,0\,0{\rightarrow}0 & 1\,0\,0\,0{\rightarrow}0 & 0\,0\,0\,0{\rightarrow}1 & 0\,0\,1{\rightarrow}1 \end{array} \tag{9}$$

The implicants in (9) are prime that is they are the initial implicants and are all in reduced form. This can be seen by reducing the antecedent by a single symbol on the left, making the antecedent then the right most subset (suffix) of another antecedent. Doing so would also remove determinism from the collection of prime implicants. This deterministic behavior is important for this paper, but non-determinism can also be important for other applications.

This structure in (9) is called a ruling, and represents the set of implicants necessary for regeneration of the original sequence. We also point out for this example: it is possible to reduce the MIB to three for the implicants in (9). All of the implicants of (9) are in a ruling with one level. As we reduce the MIB to some value less than four for this sequence, the number of levels will increase with an upper bound on the levels as well as the implicants equal to the number of symbols in one period of the sequence. In the worst case, there will be one implicant in each level of the ruling. The process of reducing the MIB requires marking in the sequence all consequents that come from implicants whose antecedents do not meet the criteria for the new MIB value of three. Those symbols marked get pushed out to the next level and that level is then factored into its prime implicants. The process continues until there are no more symbols left to be pushed out in the sequence of the highest level.

Fig. 4 shows this process with the marked symbols underlined in Level 0 getting pushed out to Level 1. Level 1 is then factored independently of Level 0, and the only implicants remaining in Level 0 are those where the antecedents meet the a-priori, MIB criteria.

Level 1: 1 1 0 || 1 Implicants: 1 1→0 0→1
Level 0: $\underline{1}$ $\underline{1}$ 0 1 0 0 0 $\underline{0}$ || 1 $\underline{1}$ Implicants: 1 1→0 1 1 0→1 1 0 1→0
0 1 0→0 1 0 0→0 0 0 1→1

Fig. 4. Implicants in a ruling of two levels with an MIB of three or less from (8).

Level 5: $\underline{1}$ $\underline{1}$ $\underline{0}$ || $\underline{1}$ $\underline{1}$ Implicants: 1 1→0 1 0→1 0 1→1
Level 4: $\underline{1}$ 1 1 0 || $\underline{1}$ 1 1 Implicants: 1 0→1
Level 3: 1 1 1 0 $\underline{0}$ || 1 1 1 Implicants: 1 0→0
Level 2: 1 1 1 0 $\underline{0}$ 0 || 1 1 1 Implicants: 1 0→0
Level 1: 1 1 1 0 $\underline{0}$ 0 0 || 1 1 1 Implicants: 1 0→0
Level 0: 1 1 $\underline{0}$ 1 0 0 0 0 || 1 1 $\underline{0}$ 1 Implicants: 1 1→0

Fig. 5. Implicants in a ruling of six levels with an IB of two from (8).

As seen from Fig. 4, the number of implicants is O(m) where m is the length of the sequence which is eight. We also see the MIB for the ruling is now three. Fig. 5 shows the ruling for an MIB of 2. From (9), Figs. 4 and 5, the number of levels increases from 1 to 2 to 6. In [6] we showed that if the IB of any of the prime implicants in Level 0 has a value less than the MIB for that level, then we can reduce the MIB to some number smaller than the prime implicant MIB, and not less than the smallest IB in level 0.

Now consider the second label sequence in (7) which is 1 1 1 0 0 0 1 0. We factor this sequence, obtaining the prime implicants in (10). All of the implicants have an IB equal to three. There is no possible reduction in the antecedents for this kind of sequence.

$$1\,1\,1\rightarrow 0 \quad 1\,1\,0\rightarrow 0 \quad 1\,0\,0\rightarrow 0 \quad 0\,0\,0\rightarrow 1 \quad 0\,0\,1\rightarrow 0 \quad 0\,1\,0\rightarrow 1 \quad 1\,0\,1\rightarrow 1 \quad 0\,1\,1\rightarrow 1 \tag{10}$$

We also note that this sequence has particular characteristics:

- The number of zero's and one's are perfectly equal.
- There is no run of zero's or one's that is longer than the MIB.
- The runs of any shorter subsequences are balanced.
- The autocorrelation function with lag equal to zero is maximum.
- The length of the sequence is 2^n with $n = 3$.

If a sequence has an unequal number of 1's and 0's, then the sequence will always produce a ruling whose individual IB's will not be equal for all of the prime implicants in Level 0. It is possible to reduce the MIB. Every label sequence fitting the five conditions above is called an FI random sequence.

4. Digraph Generation

The question becomes how to find these FI random sequences and are they really random? We address the first part of this question in this section. From the example in Section 2 resulting in (7), we saw that one of the Hamiltonian cycles was FI random and the other was not. We now construct a digraph where every Hamiltonian cycle produces an FI random sequence. In this paper we only deal with binary sequences, and in the future we will generalize this to any sequence generated by symbols from any number system of base b.

4.1 Label Sequences Base 2

Consider the vertices of a digraph labeled with 0 through 7. We connect these vertices to other vertices by S and SC operation to generate the edges. Table 1 shows the relationship for the eight vertices. This table shows the original vertex and the associated two vertices.

Given the values in Table 1, we can now connect the vertices in the digraph. This was shown in Fig. 1 which is the deBruijn graph for 8 vertices. We reproduce and augment the graph of Fig. 1 in Fig. 6 with the label edges: 1 designating a down direction while 0 indicates an up or cross direction at the same level.

Given a Hamiltonian cycle, the edge-label sequence can be constructed. Consider the cycle given in (11).

$$000\rightarrow 001\rightarrow 011\rightarrow 111\rightarrow 110\rightarrow 101\rightarrow 010\rightarrow 100\rightarrow 000 \tag{11}$$

Constructing the edge-label sequence comes by labeling the '→'s between the vertices with the edge labels in the digraph. This sequence is given in (12).

$$1\,1\,1\,0\,0\,0\,1\,0 \tag{12}$$

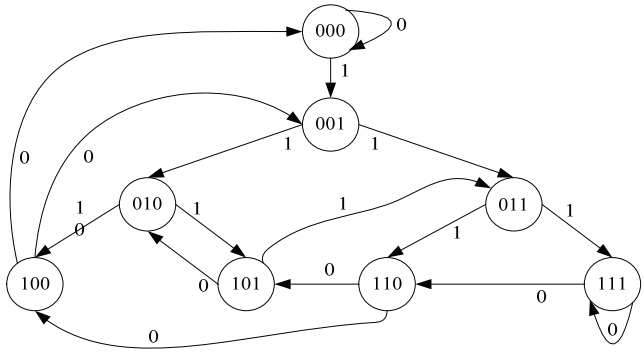

Fig. 6. Digraph formed from Table 1 with edges labeled.

The only other Hamiltonian cycle for the digraph of Fig. 6 is given in (13) together with its edge-label sequence in (14).

$$000\rightarrow001\rightarrow010\rightarrow101\rightarrow011\rightarrow111\rightarrow110\rightarrow100\rightarrow000 \tag{13}$$

$$1\ 1\ 1\ 0\ 1\ 0\ 0\ 0 \tag{14}$$

We note that the sequence in (12) is identical to that given in (10) from the non-deBruijn graph. The reason for this deBruijn sequence in a non-deBruijn digraph is the graph in Fig. 3 shares a common path with that in Fig. 6. Thus, the intersection of Fig. 6 and Fig. 3 of interest is the sequence in (12).

Table 2. Sequence of vertices given in (13) and associated bit sequences from bit positions in the vertices

		First bit	Second bit	Third bit
001	SC	0	0	1
010	S	0	1	0
101	SC	1	0	1
011	S	0	1	1
111	SC	1	1	1
110	SC	1	1	0
100	SC	1	0	0
000	SC	0	0	0

Table 3. Dual sequence of vertices from Table 2 and their associated bit sequences from bit positions in the vertices

		First bit	Second bit	Third bit
001	SC	0	0	1
010	S	0	1	1
101	SC	1	1	1
011	S	1	1	0
111	SC	1	0	1
110	SC	0	1	0
100	SC	1	0	0
000	SC	0	0	0

4.2 Hamiltonian Cycles from deBruijn Graphs

If we arrange (13) in a column in the order the vertices were generated, then we can observe the following patterns shown in Table 2 where the sequence of vertices can be represented as S or SC. As is seen in Table 2, each of the bit pattern sequences is identical to the edge sequence of (12) except they are circularly shifted. One can factor any of the sequences and obtain the original vertices, in shift order, unless the bit pattern is shifted first, as given in Table 2. That is every bit sequence in Table 2 is likewise FI random.

We define another vertex sequence called the Dual. A Dual sequence is generated by reversing the S/SC sequence and starting at vertex 000, generating another set of vertices. If we reverse the sequence of S and SC given in Table 2, we generate Table 3.

The bit sequences of Table 3 represent the edge sequence of (14) again circularly shifted. And just like the sequences of Table 2, the MIB for any of the bit sequences will produce the vertices in Table 3, except again shifted. We also note again that the vertex sequence of Table 2 and Table 3 represent the only two Hamiltonian Cycles (HC) for the deBruijn graph of eight vertices.

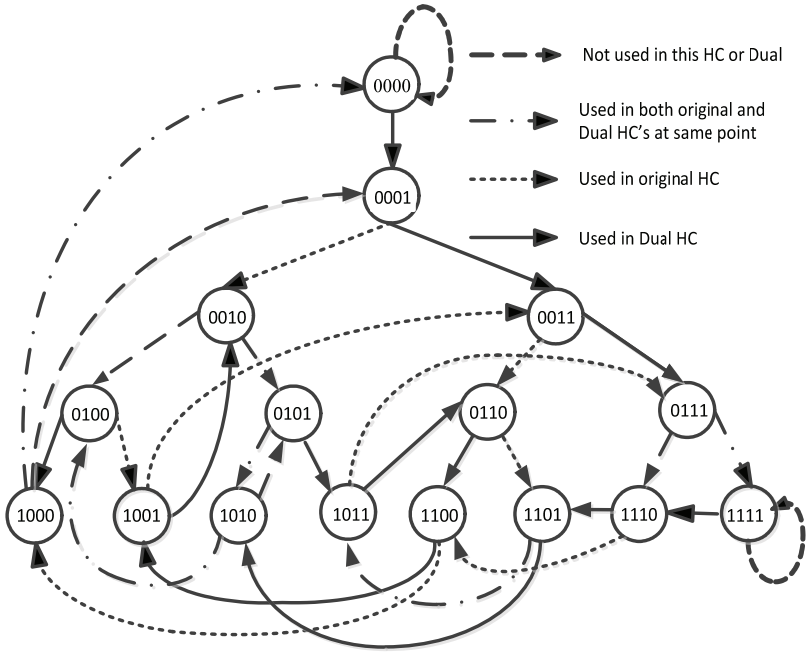

Fig. 7. Original HC in a 16 vertex graph with radius 4 and the associated, dual path.

Fig. 7 shows the deBruijn graph for 16 vertices with one HC and it's dual. From Fig. 7, it is clear that there can never be a shift, S, after vertex 0000 and 1111, rather there must always be a shift complement, SC, since a shift generates the same vertex identifier. After vertex 0111, there can likewise only be a SC, since anything else would cause vertex 1111 to be skipped, and there is no other way to reach this vertex except with a SC coming from vertex 0111. The edges from vertex 1011 to vertex 0011 and from vertex 1110 to vertex 1100 are not used, but they will be used in other HC's. The edge from vertex 1000 to vertex 0001 will likewise not be used for the same reason given for vertex 1111. Fig. 8 shows Fig. 7 with

the useless edges removed; otherwise it is identical to Fig. 7.

From these observations we can say that the every HC will contain the following vertex ordering according to the rules given in the following, if we always start at vertex 0000.

- The initial pair of vertices in the HC will always be 0000 then 0001.
- The last pair of vertices in the HC will always be 1000 then 0000.
- The HC will contain at some point, the sequence of vertices 0111, 1111, 1110 in that order.

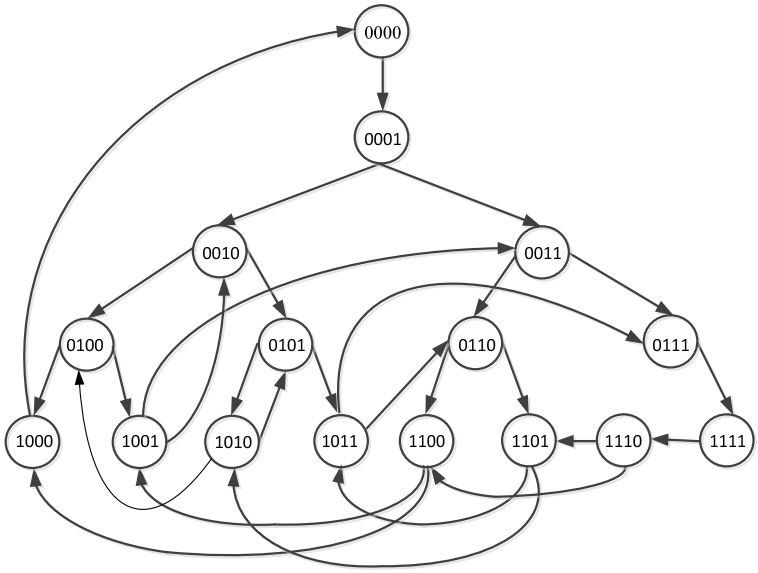

Fig. 8. Useless edges removed from graph of Fig. 7.

It should be clear that every HC has a dual, and every dual is a HC since taking the dual of the dual results in the original set of vertices which make up a HC. We can also state that the bit positions of any HC as well as its dual are simply circular shifts of the previous bit positions, and all such bit position sequences are FI random. That is, their MIB is b where 2^b is the total number of vertices in the deBruijn graph. This indicates that the factoring of the resulting bit sequence cannot have its MIB reduced. This also implies the number of levels in the ruling generated by the factoring of such sequences will be one. Further, since the number of 0's and 1's are equal in every string so derived, the Shannon entropy will be maximum for these FI random strings.

4.3 Runs in FI Random Sequences

Consider the following arbitrary examples of sequences in (15), (16), (17), and (18) produced from an 8, 16, 32 and 64 vertex deBruijn graph respectively.

11101000 (15)

1111010011000010 (16)

11111000001000110010100111010110 (17)

1111110111100111010111000110110100110010110000101010001001000000 (18)

Examining the runs we see that there are two runs of length b where again b is 2^b and for these sequences, b is 3, 4, 5 and 6. We have characterized the runs in Table 4 based upon b.

Table 4. Run counts for FI random sequences from HC of deBruijn graphs

	Runs=6	Runs=5	Runs=4	Runs=3	Runs=2	Runs=1
b=3	0	0	0	2	0	2
b=4	0	0	2	0	2	4
b=5	0	2	0	2	4	8
b=6	2	0	2	4	8	16

As can be seen from Table 4 and the properties for LFSR's as defined by Golomb [12], these sequences do not have runs corresponding to those defined in [12]. It is also easy to see the general pattern for runs in the FI random sequences for any value of b from Table 4. The last condition for randomness was the autocorrelation function must be equal to the number of 1's in the sequence when the lag is zero [12]. As mentioned, this is the case for FI random sequences.

4.3.1 Constructing FI Random Sequences

Considering the values in Table 4, we can construct candidate FI random sequences rapidly by simply combining the subsequences required in any order, just alternating the 0 and 1 subsequences. For example, if we consider b=4, and using the HC sequences from Fig. 7, we have the building blocks shown in Table 5. The Identifier B (block) is followed by the type block, a 0 or a 1, and then the length of the block. B04 means Block of length 4 of zeros.

Table 5. Building blocks identified for b = 4 and Fig. 7 from Table 4

	Length 4	Length 2	Length 1
Block	B04: 0000	B02: 00	B01: 0
	B14: 1111	B01: 11	B11: 1

Now we can form a path to build candidate FI random HC's. This is shown in Fig. 8. In this graph, one can start at any block, and then follow any edge to another vertex, and then to the next until each vertex has been visited exactly once. This would be a modified HC in that we do not return to the starting point.

From Fig. 9, the number of potential paths through this graph yields the number of paths as shown in (19) if we always start in B14 for example:

$$1*4*3*3*2*2*1*1 = 144 \tag{19}$$

In actuality there are only 16 FI random cycles for the graph in Fig. 7 not counting the rotational shifts for each of the basic cycles (i.e. those that always begin at vertex 0000). If we allow any starting vertex, then the number becomes 1152. If we consider all circular rotations of the 16 sequences, then we have a total of 256 such sequences. Any of the 1152 possible sequences are FI random, but there is a great deal of redundancy in computing FI random sequences in this manner. The redundancy is high using this technique, but it is also very fast to construct candidate random cycles.

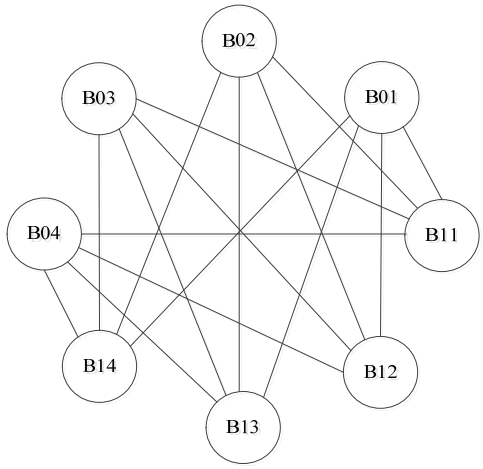

Fig. 9. Graph of possible edges between blocks to construct an FI random HC.

4.4 FI Random Considerations of LFSR Sequences

Sequences coming from LFSR are not FI random, as the number of 0's and 1's do not agree, since an LFSR does not allow a zero state. From (5) we have again the output of an LFSR as: 1 1 0 0 1 0 1. Factoring the string (20) which is just string (5) with multiple periods, where a period is separated by a vertical bar '||' we obtain the ruling for the prime implicants shown in (21).

$$1\,1\,0\,0\,1\,0\,1 \,||\, 1\,1\,0\,0\,1\,0\,1 \,||\, \ldots \tag{20}$$

$$111 \rightarrow 0 \quad 00 \rightarrow 1 \quad 010 \rightarrow 1 \quad 011 \rightarrow 1 \quad 110 \rightarrow 0 \quad 001 \rightarrow 0 \quad 101 \rightarrow 1 \tag{21}$$

Since there is an implicant in (21) with an IB less than the other individual IB's, we can reduce this set of implicants to a multi-level ruling with the MIB=2. This is shown in Table 6.

Table 6. Reduced implicants for input string (20)

Input string: 1 1 0 0 1 0 1 | 1 1 0 0 1 0 1 | …		
Level 0	Level 1	Level 2
0 0 → 1	0 1 → 1	0 1 → 1
	1 0 → 0	1 1 → 0
		1 0 → 0
		0 0 → 1

Considering the complexity of the two representations from (10) (from a FI random string) and Table 6 (from the LSFR of (5)), there are several ways to represent the complexity of the two strings. First we can say that given any pair of symbols in Table 6, we can derive the input sequence. Suppose we use the pair 01 of symbols in Table 6. In order to derive the sequence used to generate (10) it takes three symbols, and these can be again any of the symbols, say 111 for example. Thus, we can say that the length of the starting sequence required to regenerate the original sequence is a complexity measure. The more symbols required, the more complex the string. Another measure of complexity could be the

number of symbols in the ruling representing the factored sequence. As the number of symbols approaches the number of implicants times the MIB value plus one (accommodating the consequent symbol), the bigger the number, and the higher level of complexity. In the case of Table 6 this number would be 21, and in the FI random sequence from (10), the complexity number would be 32. Again the sequence (10) would be more complex.

For the two rulings in Table 6 and (10) the Kolmogorov complexity [13] would be the length of the programs necessary to regenerate the original strings. In this case, the program, given a ruling as input would be identical for any sequence of symbols. Since the program code is identical for any sequence, then the only differing measure between strings would be the ruling size. So if a finite (every finite sequence is finitely inductive (FI)) sequence or an infinite sequence (which is eventually periodic) are reduced to a ruling, then the ruling would in this domain provides the minimal program necessary to regenerate the original string. Thus, the Kolmogorov complexity would simply reduce to the number of symbols in the ruling. It is always the case that rulings will be longer than the original string, so in this sense we violate the constraint that the representation of the string must be no longer than the string. However, it is of some interest to examine the texture and complexity of strings by their ruling representation.

4.5 Finding a Cycle of an FI Random Sequence

Factoring an incoming sequence can be used to find a cycle of the sequence. Consider the sequence in (20) which is not FI random. It can be factored as shown in Fig. 10 where one symbol is input at a time.

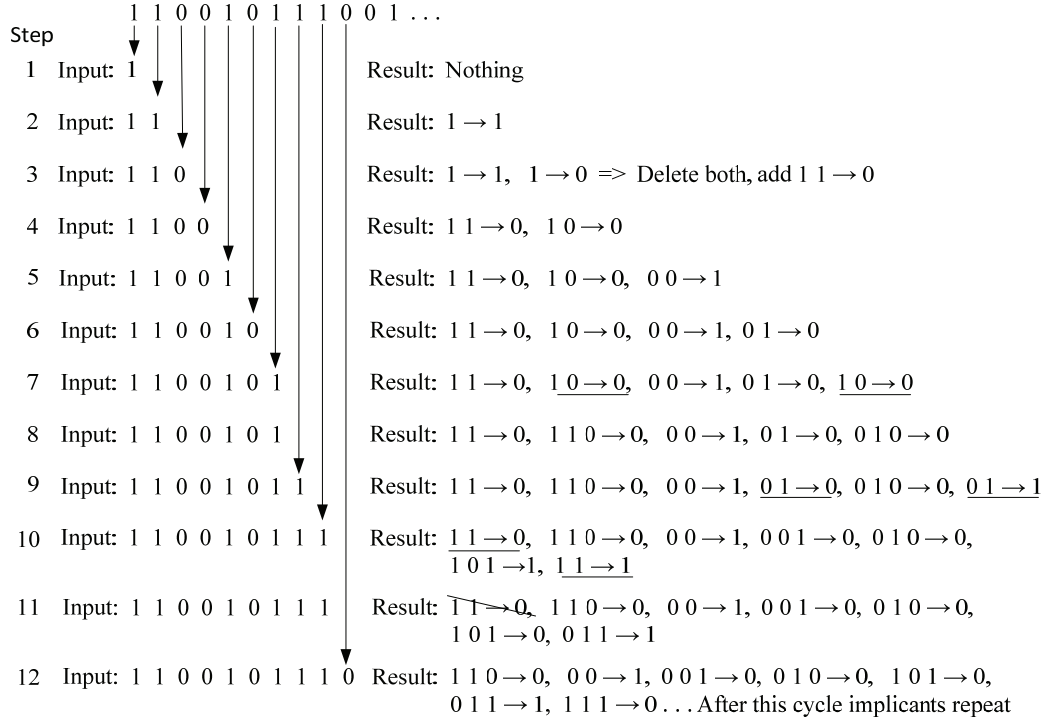

Fig. 10. Identifying a cycle from an unknown sequence one symbol at a time.

From Fig. 10, the final ruling is given in step 12, and consists of 7 implicants with MIB equal to 3. Since there is an implicant with IB=2, then the results shown in Table 6 can be obtained. In Step 3, there is a conflict that arises and as such requires both implicants found thus far to be discarded and a single new implicant defined. Doing this leaves the first two symbols of the sequence without implicants. When the sequence recycles, these two symbols will be picked up.

Steps 7, 9, 10, and 11 each require a modification to the implicants due to a conflict arising from the latest implicant and an implicant previously recorded. This is shown in the underlined implicants and in Step 11, one implicant was discarded again from a conflict.

We used this example to show that it is possible to determine the rules for a cycle in just the number of symbols in one cycle plus MIB. Once the implicants begin to repeat, a cycle has been determined. They repeat after the Step 12. This processes the last implicant in the ruling located at position cycle plus 3 and is the 10th position. The cycle has length 7 with the first 7 characters in the sequence making up one cycle.

5. Verification of Randomness

The last issue is to briefly examine the FI generated strings to determine if they are truly pseudo random sequences. In order to do this, we use some of the tests suggested in [13]. In this publication, the authors suggest a battery of 16 tests. We have not executed all of them, but show the results for a few. The tests from [13] are:

1. The Frequency Test,
2. Frequency Test within a Block,
3. The Runs Test,
4. Test for the Longest-Run-of-Ones in a Block,
5. The Binary Matrix Rank Test,
6. The Discrete Fourier Transform (Spectral) Test,
7. The Non-overlapping Template Matching Test,
8. The Overlapping Template Matching Test,
9. Maurer's "Universal Statistical" Test,
10. The Lempel-Ziv Compression Test,
11. The Linear Complexity Test,
12. The Serial Test,
13. The Approximate Entropy Test,
14. The Cumulative Sums (Cusums) Test,
15. The Random Excursions Test, and
16. The Random Excursions Variant Test.

5.1 Frequency Test

The authors suggest that the first test any proposed pseudo random sequence must pass is the Frequency Test. Here the objective is to examine the frequency of the number of 0 and 1 bits. The test

requires that one count in the sequence the excess bits of 0's or 1's over parity. Since FI random strings are both equal, then the excess turns out to be zero. They then compute:

$$S_{obs} = \frac{excess}{\sqrt{n}}, \tag{22}$$

where n is the length of the sequence. If we consider the FI random string given in (23) then the excess of either symbol will be 0 since the number of 0's and 1's are identical and both are 64.

$$\begin{array}{l} 11111110111110011110101111000111011011101001110010111000011011 00 \\ 11010101101000110010011000101100000101010010100001000100100000000 \end{array} \tag{23}$$

For (22) the $S_{obs} = 0$ for any length of sequence. They next compute the complementary error function given by the formula:

$$P\text{-}value = erfc(z) = \frac{2}{\sqrt{\pi}} \int_{z}^{\infty} e^{-u^2} du \tag{24}$$

Computing this value from [13], the value turns out to be 1, and will be 1 for all FI random sequences since z will always be 0. Since the *P-value* is > .01, we conclude that the sequence is random.

5.2 Frequency Test within a Block

The second test deals with the frequency of 0's and 1's in a block of size of length M. For the test sequence we will use the sequence in (22) and set $M = 7$, the worst possible length since one block will have all 1's in it. So the blocks from (22) are as shown as follows:

1111111	0111110	0111101
0111100	0111011	0111010
0111001	0111000	0110110
0110101	0110100	0110010
0110001	0110000	0101010
0101000	0100010	0100000

We next compute the number of 1's in each block and these are by column, 7, 5, 5, 4, 5, 4, 4, 3, 4, 4, 3, 3, 3, 2, 3, 2, 2, 1. Dividing each value by the length of the block we obtain the values which they define as π_i. These values are used in the χ^2 test to measure as described in [13] "how well the observed proportion of ones within a given M-bit block match the expected proportion (1/2)." The incomplete gamma function (igamc) is used to compute the *P-value* using the formula:

$$P\text{-}value = \text{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2}\right) \tag{25}$$

Using again the computation for the incomplete gamma function in [13], we obtain a value of .9975 where χ^2(obs) is given by the formula in [13] as:

$$\chi^2(\text{obs}) = 4M\sum_{i=1}^{N}\left(\pi_i - \frac{1}{2}\right)^2 \tag{26}$$

If the *P-value* is < .01 then the sequence is non-random. Since this is not the case, one can conclude that the appearance of 1's in each block is random.

5.3 The Runs Test

The purpose of the runs test is to determine if the oscillation between 0's and 1's is outside of the expectation for a random sequence. This test determines whether the frequency of changes in runs is too fast or too slow for a random sequence. To test to see if the FI sequence is random in this sense we again use the sequence of (23). To apply this test we first compute the test statistic given by the following formula:

$$V_n(\text{obs}) = 1 + \sum_{k=1}^{n-1} r(k)\,, \tag{27}$$

where $r(k) = 0$ if the k and k+1st elements of the sequence are identical otherwise it is 1. For (23) the result for V_n(obs) is 64. Since there is exactly the same number of ones as zeros, we have a ratio of .5 for the number of ones in the sequence. This ratio we is defined to be x in the next step which computes the *P-value* using the complementary error function as defined in Section 5.1. To compute the *P-value* [13] provides the following formula:

$$P\text{-}value = erfc\left[\frac{|V_n(\text{obs}) - 2nx(1-x)|}{2\sqrt{2n}x(1-x)}\right] \tag{28}$$

Substituting the values in this equation, we obtain for n = 128, x = 0.5, and V_n(obs) = 64, the resultant *P-value*:

$$P\text{-}value = erfc(\tfrac{0}{8}) = 1$$

This implies that for any FI sequence the *P-value* will always be 1 using the same computation as given in Section 5.1. Since the *P-value* is greater than 0.01, the sequence (23) is random.

5.4 Test for the Longest-Run-of-Ones in a Block

Here by performing this test, the interest is in the number of 1's expected in a block for a random sequence. Only the 1's contained in a block need to be tested, as the 0's would produce the same result.

This test again uses the χ^2 test. The paper [13] establishes a measure for the longest sequence of 1's in each of the blocks, and they also suggest for a sequence of length 128 that a block size of 8 bits be used. Using (23) again as the test sequence, we have the following results for the count of the number of maximal subsequences of 1's in the 16 blocks shown in below.

1 1 1 1 1 1 1 0	1 1 1 1 1 0 0 1	1 1 1 0 1 0 1 1	1 1 0 0 0 1 1 1
0 1 1 0 1 1 1 0	1 0 0 1 1 1 0 0	1 0 1 1 1 0 0 0	0 1 1 0 1 1 0 0
1 1 0 1 0 1 0 1	1 0 1 0 0 0 1 1	0 0 1 0 0 1 1 0	0 0 1 0 1 1 0 0
0 0 0 1 0 1 0 1	0 0 1 0 1 0 0 0	0 1 0 0 1 0 0 0	1 0 0 0 0 0 0 0

In each of these blocks, we identify the longest sequence of 1's and then count the number of occurrences that sequence has in all of the blocks with the constraint that V_0 is count of sequences ≤ 1; V_1 is count of sequences=2; V_2 is count of sequences of length=3; and V_3 is the count of sequences ≥ 4. From this we obtain for V_0=4; V_1=5; V_2=5 and V_3=2. We can compute the χ^2(obs) from:

$$\chi^2(\text{obs}) = \sum_{i=1}^{K} \frac{(V_i - N\pi_i)^2}{N\pi_i}, \tag{29}$$

where K is the number of data values V_i which is 3, and N is the number of blocks which is 16. The values of π_i are given by the authors and correspond to the theoretical probabilities of each of the four classes of the V_i. These values are given as π_0=0.2148; π_1=0.3672; π_2=0.2305; and π_3=0.1875. From these values the χ^2(obs) = 1.0227. In order to obtain the desired measure incomplete gamma function is used with input parameters K/2 and χ^2(obs)/2. Using this function from [13], we obtain a *P-value* = 0.1810. Since this is larger than .01 we can again conclude that the runs of 1's in a block are random.

5.5 Compression of FI Random Sequences

The last test we applied was to compress the file of 128 bits by ZIP [13] to see if 128 bit data sequence of (22) could be compressed. The file was 128 bytes, and on disk it was stored as a .text document of 4096 bytes. Using this version of ZIP the zipped file was 10,836 bytes and stored on disk it was 12,288 bytes. We tried a sequence of 1024 bits, and on the disk it required 4096 bytes for storage of 1024 bytes. The zip file results were disk size of 4096 bytes and the actual file size after compression was 418 bytes. This is not an unexpected result, as the bit string was converted into bytes and there is a great deal of redundancy in the byte representation.

The longer sequence of 1024 bits expanded by a factor of 4 to 4096 bytes before storing it in a text file. The compression then reduced these 4096 bytes to 418 bytes, or 3344 bits. If the ratio of expansion holds, then the 3344 bits would be over size by a factor of 4 making the final file size equal to 836 bits, or a compression ratio of 1.22. Since these results are just approximations, it is impossible to definitively state that there was or was not compression of the longer sequence of 1024 bits. If these FI sequences are random, then one would expect no compression, but they do contain bit sequences of up to ten 0's and 1's.

6. Conclusions

We have described a new collection of sequences coming from the deBruijn graphs for binary numbers. These new sequences we have characterized as FI random sequences. We have described how the deBruijn sequences are related to linear feedback systems, and shown that under the FI sequence definition, these derived sequences are not random because they allow one more 1 in the sequence than 0's. This allows the sequence to be factored in the FI sense using an IB less than the defined MIB.

Using the HC of the deBruijn graphs, we have generated a sequence of shifts and shift complements starting with the initial vertex 00...0 that generates each of the appropriate vertices in the HC. From this we have shown there is a dual sequence of shifts and shift complements that is the original sequence just reversed. The cycle generated from this dual is also a HC, and it and its original are both FI random. All of the FI sequences likewise have maximal entropy, unlike the sequences from the LFSR's which do not have maximal entropy.

We have also shown how the pattern established for FI random sequences generates runs of zero's and one's and how knowing this pattern allows direct generation of any FI random sequence of any length (where length is a power of 2). This process would simplify the hardware necessary for the LFSR computation, by allowing just a lookup function using a random pattern of selection of the necessary building blocks. This technique essentially provides a building block assembly process for generating such FI random sequences.

We also described how FI can be used to determine the cycle of a LFSR or a deBruijn sequence. The technique is to feed a bit sequence into the FI factoring program and wait until all of the implicants have a uniform MIB. The length of the sequence needed to determine its cycle is just the actual cycle length.

We concluded this paper with a few of the tests suggested in a report generated by Booz Allen & Hamilton for determining if a sequence is pseudo random. We applied the first five of sixteen such tests, to show that even with the zero that LFSR's leave out, the FI random sequences are indeed random. This supposes that these tests for randomness actually determine if a sequence is random. However as the authors of this document point out that these statistical tests are only a first step in determining if a particular pseudo random number generator actually generates random numbers. Because the purpose of this paper was only to further explore bit sequences generated under special conditions, and not to produce a sequence suitable for cryptographic applications, we did not include results from all proposed tests.

There is an immediate generalization of the results of this paper in numbers systems of various bases, and we have now only an unpublished version of this work. Lastly, such sequences have provided fascination for many people over the time span of their invention, and we must also conclude we too are fascinated by the regularities or perhaps better stated the irregularities of these most interesting binary sequences.

References

[1] J. von Neumann, *Collected Works*. Oxford, UK: Pergamon Press, 1962.

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 623-656, 1948.

[3] C. Stroud, *Linear Feedback Shift Registers (LFSRs)*. Auburn, AL: Department of Electrical and Computer Engineering, Auburn University, 2004.

[4] I. J. Good, "Normal recurring decimals," *Journal of the London Mathematical Society*, vol. s1-21, no. 3, pp. 167-169, 1946.

[5] N. G. de Bruijn, "A combinatorial problem," in *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen*, Amsterdam, 1946, pp. 758-764.

[6] J. H. Case and P. S. Fisher, "Long-term memory modules," *Bulletin of Mathematical Biology*, vol. 46, no. 2, pp. 295-326, 1984.

[7] R. G. Busacker and T. L. Saaty, *Finite Graphs and Networks: An Introduction with Applications*. New York, NY: McGraw-Hill, 1965.

[8] J. H. van Lint, *Combinatorial Theory Seminar, Eidhoven University of Technology*. Berlin, Germany: Springer, 1974.

[9] A. Alhakim and M. Akinwande, "A recursive construction of nonbinary de Bruijn sequences," *Designs, Codes and Cryptography*, vol. 60, no. 2, pp. 155-169, 2011.

[10] F. M. Malyshev and V. E. Tarakanov, "Generalized de Bruijn graphs," *Mathematical Notes*, vol. 62, no. 4, pp. 449-456, 1997.

[11] W. T. Tutte, *Graph Theory*. Cambridge, UK: Cambridge University Press, 2001.

[12] S. W. Golomb, *Shift Register Sequences*. Laguna Hills, CA: Aegean Park Press, 1982.

[13] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*. New York, NY: Springer, 2008.

Paul Fisher

He received the B.A. and M.A. degrees in mathematics from University of Utah, Salt Lake City, and the Ph.D. degree in computer science from Arizona State University, Tempe. He is R. J. Reynolds distinguished Professor of Computer Science at the Winston-Salem State University (WSSU), Winston-Salem, NC. He is the director of High Performance Computing Group at the WSSU. He has written and managed more than 100 proposal efforts for corporations and DoD involving teams of 1 to 15 people. He worked as a consultant to the US Army, US Navy, US Air Force, and several companies over the years. In the 1990s, he commercialized an SBIR-funded effort and built Lightning Strike, a wavelet compression codec, and then sold the company to return to academe. His current research interests include wired/wireless communication protocols, image processing, and pattern recognition.

Nawaf Aljohani

He received M.S. degree in Computer Science from Winston-Salem State University in 2014. His current research interests include pattern recognition and image processing. He is currently working at the public administration in Saudi Arabia.

Jinsuk Baek https://orcid.org/0000-0003-4848-4772

He received the B.S. and M.S. degrees in computer science and engineering from Hankuk University of Foreign Studies (HUFS), Seoul, Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science from the University of Houston (UH), Houston, TX, in 2004. He is a full Professor of computer science at the Winston-Salem State University (WSSU), Winston-Salem, NC. He is the director of Network Protocols Group at the WSSU. His research interests include wireless sensor networks, multicast protocols, mobile computing, network security, and multimedia communications. Dr. Baek has served and is serving as an active member for various technical activities in the computer network and communication field. Currently, he is serving as an editor of the *KSII TIIS Journal*, the technical committee of many leading international conferences, and a reviewer for leading journals, including *Wiley Wireless Communications and Mobile Computing, IEEE Transactions on Mobile Computing, IEEE Transactions on Broadcasting*, and *Journal of Computer Systems, Network, and Communications*.