Revista de Matemática: Teoría y Aplicaciones
ISSN: 1409-2433
mta.cimpa@ucr.ac.cr
Universidad de Costa Rica
Costa Rica

Skliar, Osvaldo; Monge, Ricardo E.; Medina, Víctor; Gapper, Sherry; Oviedo, Guillermo

A hybrid random number generator (hrng)

Revista de Matemática: Teoría y Aplicaciones, vol. 18, núm. 2, 2011, pp. 265-297
Universidad de Costa Rica
San José, Costa Rica

Revista de Matemática: Teoría y Aplicaciones 2011 **18**(2) : 265–297
cimpa – ucr issn: 1409-2433

# A HYBRID RANDOM NUMBER GENERATOR (HRNG)

# UN GENERADOR HÍBRIDO DE NÚMEROS ALEATORIOS

Osvaldo Skliar[*] Ricardo E. Monge[†]
Víctor Medina[‡] Sherry Gapper[§]
Guillermo Oviedo[¶]

*Received: 23 Feb 2010; Revised: 19 Nov 2010; Accepted: 1 Feb 2011*

[*] Escuela de Informática, Universidad Nacional, Heredia, Costa Rica. E-Mail: oskliar@costarricense.cr

[†] Universidad Interamericana, Heredia, Costa Rica. E-Mail: rmongeg@uinteramericana.edu

[‡] Escuela de Matemática, Universidad Nacional, Heredia, Costa Rica. E-Mail: vmedinabaron@yahoo.es

[§] Universidad Nacional, Heredia, Costa Rica. E-mail: sgapper@una.ac.cr

[¶] Universidad Latinoamericana de Ciencia y Tecnología, San José, Costa Rica. E-Mail: oviedogmo@gmail.com

## Abstract

The purpose of this paper is to present a novel Hybrid Random Number Generator (HRNG). Here "hybrid" refers to the fact that to construct this generator it is necessary to use 1) physical components – texts – and a physical process, and 2) a mathematical procedure. This HRNG makes it possible to generate genuine random numbers which may be used both for computer simulation of probabilistic systems and in the field of cryptography. The results of a comparative study of the binary strings generated by this HRNG and of those generated by two highly used implementations of a congruential algorithm designed to generate pseudorandom numbers are given here. One of the latter is the implementation incorporated into the Java 2 platform (version 1.6), and the other is the implementation incorporated into the runtime library of Microsoft's Visual C++ 2008 compiler.

**Keywords:** random number generator, pseudorandom number generator, hybrid random number generator.

## Resumen

Se presenta un generador híbrido de números aleatorios que será denominado, de manera abreviada, "HRNG". Mediante el calificativo "híbrido" se hace referencia al hecho de que la construcción de dicho generador requiere recurrir a 1) unos entes de carácter físico —textos— y un procedimiento físico y a 2) un procedimiento matemático. El HRNG permite generar genuinos números aleatorios que pueden ser utilizados tanto para la simulación computacional de sistemas probabilísticos como en el campo de la criptografía. Se aporta los resultados de un estudio comparativo de cadenas binarias generadas con el HRNG y cadenas binarias generadas por dos implementaciones —ampliamente utilizadas— de un algoritmo congruencial diseñado para generar números pseudoaleatorios: a) la implementación incorporada a la versión 1.6 de la plataforma Java 2 y b) la implementación incorporada a la biblioteca de ejecución del compilador Microsoft Visual C++ 2008.

**Palabras clave:** generador de números aleatorios, generador de números pseudoaleatorios, generador híbrido de números aleatorios

**Mathematics Subject Classification:** 11K45, 65C10.

# 1 Introduction

## 1.1 Main issues addressed in scientific literature on random numbers and pseudorandom numbers

In scientific literature on random numbers and pseudorandom numbers, one of the main topics is the difference between genuine random numbers and others which are purportedly similar to them but not the same. The latter are usually referred to as pseudorandom numbers.

One widely accepted criterion is that purely mathematical procedures cannot generate true random numbers. John von Neumann's famous remark may have helped to spread this opinion. He once stated jokingly: "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin" [13]. If this is accepted, then true random numbers may be generated only with certain procedures involving processes found in Physics and other related fields.

In spite of the above, the main theoretical approaches developed to characterize the notion of randomness of certain numbers are independent of how those numbers are generated [1, 4, 8, 10, 12].

Another of the important topics considered in the scientific literature is that of tests for randomness. If one has certain numbers (to be characterized preliminarily in section 1.2, and more precisely, in section 2), it is usually accepted that diverse tests may be carried out to assess whether they are random, or in any case, whether they are "random enough", according to the purpose for which they will be used [5, 7]. These initially general or somewhat vague notions will be further clarified as this paper progresses.

## 1.2 An ideal random-bit generator (IRBG) and some possible applications

For a clearer understanding of the methods presented here, it is useful to consider an ideal machine that every so often (for instance, every millisecond) generates a digit with the following two characteristics: a) the digit is either a zero (0) or a one (1), and b) there is the same probability that the digit will be a 0 or a 1 (that is, for each digit generated by this ideal machine, the probability that it could be a 1 is 0.5, and that it could be a 0 is 0.5). This ideal machine will be called an "ideal random-bit generator" (IRBG). It will also be supposed that it has an ideal "memory" device (e.g., an "ideal hard disk") which makes it possible to store a sequence of bits (generated by the IRBG) that is as long, or that contains as many bits, as desired. This sequence will be called a “binary string”.

One possible application of the IRBG is that of obtaining random numbers similar to those provided in a well-known publication of the Rand Corporation [9]. Let us admit, for example, that a sequence of five digits is required such that each of them can be one of the following: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. It can also be accepted that the probability of any one of these digits being in any of the five positions possible in any of the sequences of five digits is the same. (Hence, the probability is equal to 0.1.)

How could this type of five-digit sequence be obtained with an IRBG? The first four digits of a binary sequence generated by the IRBG and stored in a memory device will be used for this purpose. This four-digit binary sequence is interpreted first as a number expressed in base 2, and then, as a number in base 10. Thus there will be the same probability ($\frac{1}{16}$) that the binary sequence will correspond to any of these 16 numbers: $0, 1, 2, \ldots, 15$. (For example, the binary sequences 0000, 0101, 1011 and 1111 correspond to the numbers 0, 5, 11 and 15, respectively.) If the binary sequence corresponds to one of the numbers $0, 1, 2, \ldots, 9$, then the number obtained is considered the first of the numbers in the sequence of five numbers to be constructed. If, on the other hand, the four-digit binary sequence corresponds to one of the numbers 10, 11, 12, 13, 14 or 15, it is discarded, and the next four digits of the binary string mentioned above (generated by the IRBG and stored in the memory device) are considered. The same procedure is carried out with that sequence of four more new digits as was done with the first four-digit binary sequence. One continues to repeat the procedure until obtaining the desired sequence of five digits expressed in base 10. It is evident that each of these five digits has the same probability (0.1) of being any of the following ten numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

This procedure can be used as many times as needed to obtain the required random numbers. Although in the characterization of this procedure it was taken into account that the numbers obtained are expressed in sequences of five digits corresponding to base 10, clearly the same approach can be applied to obtain random numbers which can be expressed in sequences of a finite number of digits (corresponding to that base) that are as long as desired.

Is it possible for the number of digits in the binary string generated by the IRBG and stored in its memory device not to be long enough (e.g., in cases in which many sequences of four-digit binary sequences are discarded) to obtain all the sequences of a finite number (i.e., sequences of digits in base 10, which are as long as necessary) of digits expressed in base 10) that one wishes to construct? Not at all. It must be kept in mind that this IRBG is an ideal machine that never fails. Using this machine, one can obtain a binary string composed of a series of digits which is as long as required. Moreover, that binary string can always be stored in the ideal memory device mentioned. (What happens in the "real world"? This will be covered in section 2.) This procedure may be used, therefore, as many times as needed, with the objective of constructing as many random numbers as desired.

Another possible application of the IRBG is the generation of probabilities expressed with as many decimal digits as needed and such that each of these digits has the same probability (0.1) of being any one of the following ten numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Suppose, for instance, that it is necessary to generate a probability expressed with 8 decimal digits. To achieve this, the first step is to generate, using the procedure described above, a random number expressed by an eight-digit sequence in base 10. Let us admit that the sequence generated is 08760552. To obtain the probability desired, the sequence can be considered to be the "decimal part" of that probability. Hence, the probability obtained is 0.08760552.

### 1.3 Objective of this paper

The objective of this paper is to present a system whose behavior is the same as that of an IRBG, or at least as similar as possible. Once the system is described, an analysis will be carried out as to how similar it is to the IRBG.

## 2 First approximation to a physical-mathematical system which operates like an IRBG

At present an impressive number of literary works (such as novels and essays) are easily accessible on the Internet. One of these was selected and processed in a particular way. (All of the literary works used for this purpose were taken from "Project Gutenberg" – www.gutenberg.org.) To understand the nature and the sense of this procedure, one must keep in mind that this literary text (or any literary work, for that matter) can be considered to consist of a sequence of characters. (The term "character" is used here with the meaning of sign or symbol, as commonly found in Computer Science.) One objective of this procedure is to prevent the appearance of subsequences of characters composed of numerous repetitions of the same character such as that corresponding to a blank space (between words). Another objective is to prevent the presence of long subsequences of characters that are identical or very similar to those found in other works. (Licenses regarding the use of the texts or other legal information can be the cause of this type of subsequences of characters.) The reason for this procedure will be discussed in section 3.

The text resulting from the work subjected to this treatment can be considered a sequence of characters. The first character in the sequence will be called "character 1", the second "character 2", the third "character 3", and so on. Each of these characters is one of those specified in Table 1.

Consider the subsequence composed of the first 100 characters of the sequence of characters obtained when processing the selected literary work as specified. One of these 100 characters may be chosen using any criterion or method; that is, it is arbitrary. The character thus selected will be called $s_1$. Beginning with $s_1$, an advance of 1600 characters is made within the sequence considered in order to determine another character which will be called $s_1^*$. In other words, there are 1599 characters between $s_1$ and $s_1^*$. Metaphorically, the operation can be described as a "long leap" of 1600 characters from the initial character $s_1$ to the final character $s_1^*$. (For example, if $s_1$ was "character 57", then $s_1^*$ would be "character 1657".) Consideration is given to the order numbers in Table 1 which correspond to $s_1$ and $s_1^*$. If $s_1$ is assigned a number whose order number is less than that of $s_1^*$, then the digit 0 will be the first digit of a binary string under construction. (This binary string will be the first approximation to a hypothetical binary string generated by an IRBG.) Therefore, for example, if the order number of $s_1$ (in Table 1) is 53 and the order number of $s_1^*$ (also in Table 1) is 118, then the first digit of the binary string to be generated is 0, since $53 < 118$. On the other hand, if the order number corresponding to $s_1$ is greater than the order number of $s_1^*$, then the first digit of the binary string to be generated is 1. If $s_1$ is identical to $s_1^*$, then their order numbers are the same. In this case, it will be considered that no digit for the binary string to be constructed has been generated from $s_1$ and $s_1^*$.

Attention will now be given to another two characters from the sequence of characters to which reference was made. These two characters – $s_2$ and $s_2^*$ – are determined in the following way: An advance of 50

| Number | | Number | | Number | | Number | |
|---|---|---|---|---|---|---|---|
| 0 | (ctl) | 32 | (blank space) | 64 | @ | 96 | ‘ |
| 1 | (ctl) | 33 | ! | 65 | A | 97 | a |
| 2 | (ctl) | 34 | ” | 66 | B | 98 | b |
| 3 | (ctl) | 35 | # | 67 | C | 99 | c |
| 4 | (ctl) | 36 | $ | 68 | D | 100 | d |
| 5 | (ctl) | 37 | % | 69 | E | 101 | e |
| 6 | (ctl) | 38 | & | 70 | F | 102 | f |
| 7 | (ctl) | 39 | ’ | 71 | G | 103 | g |
| 8 | (backspace) | 40 | ( | 72 | H | 104 | h |
| 9 | (tab) | 41 | ) | 73 | I | 105 | i |
| 10 | (line feed) | 42 | * | 74 | J | 106 | j |
| 11 | (ctl) | 43 | + | 75 | K | 107 | k |
| 12 | (ctl) | 44 | , | 76 | L | 108 | l |
| 13 | (carriage return) | 45 | - | 77 | M | 109 | m |
| 14 | (ctl) | 46 | . | 78 | N | 110 | n |
| 15 | (ctl) | 47 | / | 79 | O | 111 | o |
| 16 | (ctl) | 48 | 0 | 80 | P | 112 | p |
| 17 | (ctl) | 49 | 1 | 81 | Q | 113 | q |
| 18 | (ctl) | 50 | 2 | 82 | R | 114 | r |
| 19 | (ctl) | 51 | 3 | 83 | S | 115 | s |
| 20 | (ctl) | 52 | 4 | 84 | T | 116 | t |
| 21 | (ctl) | 53 | 5 | 85 | U | 117 | u |
| 22 | (ctl) | 54 | 6 | 86 | V | 118 | v |
| 23 | (ctl) | 55 | 7 | 87 | W | 119 | w |
| 24 | (ctl) | 56 | 8 | 88 | X | 120 | x |
| 25 | (ctl) | 57 | 9 | 89 | Y | 121 | y |
| 26 | (ctl) | 58 | : | 90 | Z | 122 | z |
| 27 | (ctl) | 59 | ; | 91 | [ | 123 | { |
| 28 | (ctl) | 60 | < | 92 | \ | 124 | \| |
| 29 | (ctl) | 61 | = | 93 | ] | 125 | } |
| 30 | (ctl) | 62 | > | 94 | ˆ | 126 | ˜ |
| 31 | (ctl) | 63 | ? | 95 | _ | 127 | (ctl) |

Table 1: List of characters and order numbers. (Source: The Unicode Standard, Version 5.0, Fifth Edition, The Unicode Consortium, Addison-Wesley Professional, 27 October 2006.)

characters from $s_1$ was performed to determine the character called $s_2$. In other words, there are 49 characters between $s_1$ and $s_2$. Metaphorically, this may be described as a "short leap" of 50 characters from the initial character $s_1$ to the final character $s_2$. (For example, if $s_1$ was "character 57", then $s_2$ would be "character 107".) As of $s_2$, there is a "long leap" of 1600 characters to be able to determine another character to be called $s_2^*$. (For example, if $s_2$ was "character 107", then $s_2^*$ would be "character 1707".) Note the order numbers in Table 1 which correspond to $s_2$ and $s_2^*$. If the order number corresponding to $s_2$ is less than the order number corresponding to $s_2^*$, then the digit obtained is 0. If the order number corresponding to $s_2$ is greater than the order number corresponding to $s_2^*$, then the digit obtained is 1. If the order number of $s_2$ is identical to that of $s_2^*$, then they both will be the same character. In that case, it will be considered that from $s_2$ and $s_2^*$, no digit was obtained for the binary string to be built. If, based on the comparison of the order numbers of $s_1$ and $s_1^*$, a digit has been obtained, then that digit will be the second digit in the binary string. If, based on the comparison of order numbers corresponding to $s_1$ and $s_1^*$, no digit was obtained previously, then the digit obtained from the comparison of the order numbers corresponding to $s_2$ and $s_2^*$ will be the first digit obtained for the binary string. The same procedure will be used repeatedly to obtain the successive digits belonging to the binary string to be generated.

Another way to characterize the procedure used to generate this binary string is as follows: Beginning with $s_1$, with successive "short leaps" of 50 characters, the characters $s_2, s_3, \ldots, s_{n-1}, s_n$ are selected successively. Thus, one has the sequence of characters $s_1, s_2, s_3, \ldots, s_n$. Beginning with each of the characters comprising this sequence, there is a "long leap" of 1600 characters. Hence, from character $s_1$, character $s_1^*$ is selected, from $s_2$, character $s_2^*$ is selected, and so forth, until reaching $s_n$, from which $s_n^*$ is selected. One has then another sequence of characters: $s_1^*, s_2^*, s_3^*, \ldots, s_n^*$.

Later the order number corresponding to $s_1$ (in Table 1) is compared to the order number corresponding to $s_1^*$, that of $s_2$ to that of $s_2^*$, and so on successively. If the order number corresponding to $s_i$, for $i = 1, 2, \ldots, n$, is less than the order number corresponding to $s_i^*$, then it is considered that from the ordered set of characters $\{s_i, s_i^*\}$, the digit obtained is 0. If the order number corresponding to $s_i$ is greater than that of $s_i^*$, then it is considered that from the ordered set $\{s_i, s_i^*\}$, the digit 1 is obtained. If $s_i$ is identical to $s_i^*$ (and therefore, the order number of $s_i$ is equal to that of $s_i^*$), it is considered that from the ordered set $\{s_i, s_i^*\}$, no digit is obtained. Given that from each ordered set $\{s_i, s_i^*\}$, a digit may or may not be obtained, the number of digits will be, at most, the same as the number of ordered sets that have been considered.

Given that the digits of a binary string to be constructed are, whenever feasible, obtained sequentially from the ordered sets $\{s_1, s_1^*\}$, $\{s_2, s_2^*\}$,$\ldots$, $\{s_n, s_n^*\}$, an order can be established for those digits.

Suppose that the number of digits of the binary string generated with the procedure described turns out to be insufficient, given the objective for which the string will be used. One may select, then, another literary work. The same procedure will be applied to this second text. Thus a new binary string is obtained. This second binary string is a continuation or prolongation of the first. If necessary, a third text may be selected, from which a third binary string is obtained using the above procedure which was already used twice. This string is also considered a continuation of the binary string obtained previously. This procedure may be repeated as many times as needed. Since it may be required to use many literary texts to obtain a binary string composed of a number of bits long enough for the purpose intended, it is easy to understand why, as indicated above, the paragraphs which are the same or similar in more than one text must be eliminated. In this way, it is possible to remove from the texts any significant patterns or regularities which could interfere with the objective of obtaining a genuinely random string from those texts.

The sequence of characters obtained from all of the literary works will be called $S_1$, and the binary string obtained from it, $B_1$. Moreover, in section 3.2 a description will be provided of another procedure which will make it possible to increase the number of bits in a given binary string significantly without using additional literary works.

The theoretical basis of the procedure described to generate a binary string is as follows: Let there be a character (e.g., a letter) from a written literary text in a particular language (such as English). Suppose that from that character there is a "leap" of 50 characters or more to determine the second character. (In other words, between that first character and the second, there are at least 49 characters.) Admit that for this type of "leap", the identity of the second character does not depend on that of the first. The basis for this supposition is that between the first character and the second there are characters corresponding to at least two words, even if they are very long words made up of many letters. (Each letter is considered a character.) Suppose that the first character corresponds to the first letter of a 14-letter word. The first 13 characters between the initial character (the character from which the leap was made) and the second (the "target" character) correspond to the rest of the word mentioned. Then a character corresponding to a blank space between the words can be identified. The presence of two 14-letter words (one after another) and a space between them would require another 29 characters. Thus it is possible to justify the use of the first 42 characters of the 49 characters found between the initial character of the 50-character "leap" and the "target" character. The presence of the characters corresponding to at least two words between the "initial" character and the "target" character gives a high degree of verisimilitude to the hypothesis stated above. Hence, let there be 4 consecutive words ($w_1, w_2, w_3$ and $w_4$). Consider a "leap" beginning from any character in $w_1$ and ending on any character of $w_2$. It appears highly unlikely that by knowing a character in $w_1$ it would be possible to guess what the character in $w_2$ might be. This would occur even if all the characters in $w_1$ are known; that is, even if one knows what $w_1$ is. There are many words which may conceivably come after $w_1$, and become $w_2$. Reasoning of this type is applicable to $w_2$ and $w_3$, and also to $w_3$ and $w_4$. Consider now the "leap" going from some character in $w_1$ to some character in $w_4$. For the reasons given above, it is highly unlikely that by knowing that character in $w_1$, one could infer anything with regard to the "target" character in $w_4$. Therefore, if from a given character a "short leap" of at least 50 characters is made to a second character, it is reasonable to accept that this last character is independent of whatever the first one was.

For these same reasons, a "leap" of 1600 characters (the "long leap" mentioned above), ensures that the type of independence described is achieved between the first character and second one. Therefore, it can be admitted that regardless of what character $s_1^*$ is, it does not depend on what character $s_1$ is. (The notation introduced above is being used here.) Likewise, regardless of what character $s_2^*$ is, it does not depend on what character $s_2$ is, etc.

The first character, the second character, . . . , and the last character in Table 1 will be called $c_1, c_2, \ldots, c_{128}$. The following hypothesis will be accepted: if a character is selected at random in a literary text written in a particular language, such as English, there is a certain probability that it will be the first character on the list in Table 1, another probability that it will be the second character on the list, and so on. The probability that upon choosing a character at random in that text, the character will be $c_1$ will be called $p(c_1)$. (It may be considered that the value of that probability depends on, or is a function of, $c_1$.) Likewise, the probability that, when choosing a character at random in that text, the character will be $c_2$ will be called $p(c_2)$. In general, the probability that upon choosing a character at random in that text, that character will be $c_i$ is $p(c_i)$ for $i = 1, 2, \ldots, 128$.

Look again at the ordered sets $\{s_i, s_i^*\}$, for $i = 1, 2, \ldots, n$, mentioned above. Also consider any two characters (such as $c_{68}$ and $c_{116}$) in Table 1. Suppose that any one of the ordered sets is chosen at random. The probability that the ordered set will be the same as the ordered set $\{c_{68}, c_{116}\}$ will be specified as $p\{c_{68}, c_{116}\}$. Given that the probability that second element of the ordered set will be $c_{116}$ is independent of the fact that the first element of the ordered set is $c_{68}$, the following equation is valid:

$$p\{c_{68}, c_{116}\} = p(c_{68}) \cdot p(c_{116}). \tag{1}$$

Likewise, for the probability that the ordered set, chosen at random, will be the ordered set $\{c_{116}, c_{68}\}$, the following is valid:

$$p\{c_{116}, c_{68}\} = p(c_{116}) \cdot p(c_{68}). \tag{2}$$

Note that the second members of equations (1) and (2) are equal. (The order of the factors – two probabilities – does not alter the value of the product.) Thus, the first members of these equations are also equal:

$$p\{c_{68}, c_{116}\} = p\{c_{116}, c_{68}\}. \tag{3}$$

Whenever there is an ordered set $\{c_{68}, c_{116}\}$, in the ordered sets $\{s_i, s_i^*\}$, where $i = 1, 2, \ldots, n$, a 0 will be generated for the desired binary string; and whenever there is an ordered set $\{c_{116}, c_{68}\}$, a 1 will be generated for that binary string. According to the procedure specified for that binary string, the literary works subjected to this treatment will be converted to a sequence of characters $S_1$. Each of the characters is on the list in Table 1. Two of these characters are $c_{68}$ and $c_{116}$. It is clear that the number of characters composing $S_1$ will be well over 128. Thus, many of these 128 characters will appear many times in $S_1$. This will occur, in particular, with the characters $c_{68}$ and $c_{116}$. Some of the ordered sets $\{s_i, s_i^*\}$, where $i = 1, 2, \ldots, n$, will be sets to which the elements $c_{68}$ and $c_{116}$ belong. Consider any one of these ordered sets and admit that it is known that $c_{68}$ and $c_{116}$ belong to it, but that the order of these elements in the set is not known. With equation (3), it can be stated that the probability that from that set a 0 will be obtained as a digit of the binary string is the same as the probability that a 1 will be obtained. Considerations like those presented for $c_{68}$ and $c_{116}$ could be given for any pair of characters $c_i$ and $c_j$ (for $i = 1, 2, \ldots, 128$; and $j = 1, 2, \ldots, 128$) of those appearing on the list of characters in Table 1. Thus, equation (3) can be generalized as follows:

$$p\{c_i, c_j\} = p\{c_j, c_i\}. \tag{4}$$

Any of the ordered sets $\{s_i, s_i^*\}$, where $i = 1, 2, \ldots, n$, will be one of the ordered sets $\{c_i, c_j\}$, where $i = 1, 2, \ldots, 128$, and $j = 1, 2, \ldots, 128$. For each instance in which $i = j$, no digit (neither 0, nor 1) will be generated from the ordered set considered. For each case in which $i \neq j$, either a 0 or a 1 will be generated from that ordered set, according to the criterion specified above. The probability that the presence in the ordered sets $\{s_i, s_i^*\}$ (for $i = 1, 2, \ldots, n$) of any ordered set from which the digit 0 is generated from a binary string that is being constructed is the same (see equation (4)) as the probability of the presence of a "symmetric" set with respect to that from which a digit 1 is generated for the binary string. For this reason, the probability that any digit chosen at random from the binary string obtained will be a 0 is the same as the probability that it will be a 1. Each of those two probabilities ought to be, therefore, equal to $\frac{1}{2}$. This is precisely the defining characteristic of the binary string generated by an IRBG. The conclusions reached in this section have not been proven, but a number of reasons have been given showing that they are plausible. Using a computer program designed for the purpose, it is possible (a) to count the number of times $n\{c_i, c_j\}$ that any of the ordered pairs of characters $\{c_i, c_j\}$ of the type mentioned in equation (4) appears in the sequence of all the pairs of characters actually used to compare the order numbers corresponding to the two characters belonging to each pair; and (b) to divide that number $n\{c_i, c_j\}$ by the total number $N_T$ of pairs of characters actually used existing in that sequence of pairs of characters to which reference was made. Thus, it is possible to determine the frequency $f\{c_i, c_j\}$:

$$f\{c_i, c_j\} = \frac{n\{c_i, c_j\}}{N_T}. \tag{5}$$

With the same procedure, the frequency of $f\{c_j, c_i\}$ may be obtained. However, $f\{c_i, c_j\}$ and $f\{c_j, c_i\}$ are the best approximations for the probabilities $p\{c_i, c_j\}$ and $p\{c_j, c_i\}$ respectively. Therefore, if the main hypothesis of this paper, expressed by equation (4) – $p\{c_i, c_j\} = p\{c_j, c_i\}$ – is correct, the following equation also should be valid:

$$f\{c_i, c_j\} \simeq f\{c_j, c_i\}. \tag{6}$$

It was verified that (6) is true for each of the seven ordered pairs of characters and their corresponding symmetrical pairs of those characters. These results are presented in section 6.

In section 5, a presentation will be provided of the results obtained when subjecting to statistical tests the conclusion that the procedure described above leads to a system operating like an acceptable first approximation to an IRBG.

# 3 Second approximation to a physical-mathematical system which operates like an IRBG

The physical-mathematical system discussed in section 2 generates a binary string (namely $B_1$) which as shown in section 5, passes the statistical tests described there. Thus, it is reasonable to consider that the binary string $B_1$, generated by a first approximation to an IRBG, will be acceptable in the fields of statistics (e.g., to generate random numbers) and of computer simulation (e.g., to decide whether an event of a probabilistic nature will occur during a simulation process that is being carried out). Nevertheless, doubts could arise concerning the acceptability of that binary string in the field of cryptography, where the messages encrypted using a random binary string (like $B_1$) should be able to be deciphered only by using it again. (This issue will be addressed in section 3.2.) Some person P who is not authorized to have access to the string might obtain it, and consequently, decipher those messages, if that person made correct, detailed guesses regarding the procedure used to generate the string. For the string to be acceptable in the field of cryptography, the probability that P would be able to make those guesses must be extremely low; that is, it should have a value very close to 0.

This second approximation to a mathematical system operating like an IRBG does make it possible for the binary string generated by this second physical-mathematical system to be acceptable not only in the fields of statistics and computer simulation but also in that of cryptography [6].

## 3.1 One way to generate a random permutation with the first 128 natural numbers

A total of 128 rectangular cards were prepared. A “1” was written on one side of one of these cards, a “2” on another of these cards, a “3” on still another of these cards, and so forth, in such a way that the number

“128” was written on one side of the last of these cards. (A briefer way of expressing what was done with the cards is as follows: They were numbered from 1 to 128.) Next, with all of the numbered cards going in the same direction, they were shuffled as is often done with a pack of cards. This pack of numbered cards was placed face-down on a table. The top card was selected, and note was made of the corresponding number (i.e., of the number written on the card). That number was given the number “1”, and the card was not returned to the pack. Again the top card was chosen, note was taken of the number on the card, and it was assigned the number “2”. It was not returned to the pack either. For example, suppose that the first card had the number “17” and the second was “105”. In this case, “17” would be “1”, and “105” would be “2”, and so on successively. Thus, one permutation of the 128! possible permutations of the first 128 natural numbers was obtained.

## 3.2 Characterization of the second approximation to a physical-mathematical system which operates like an IRBG

The permutation obtained as specified in section 3.1 was applied to obtain a permutation of the characters in Table 1. Therefore, for example, suppose that the permutation obtained with that procedure assigned the numbers 1 and 2 to numbers 17 and 105, respectively. In that case, the permutation of the characters in Table 1 will be one in which $c_1$ is replaced by $c_{17}$, and $c_2$ by $c_{105}$. In the same way, each of the remaining characters in Table 1 may be replaced by some character in the same table. This permutation of the characters in Table 1 does not apply to the table; the table remains unchanged. The permutation obtained from the characters in Table 1 is applied to the entity to be specified in this section.

There will necessarily be a one-to-one correspondence between the set of characters in Table 1 and the set of characters obtained with the application of the permutation described. In other words, the permutation will be “character to character”, and not “one character to several characters” or “several characters to one”. Of course, given the nature of the permutation considered, the set of characters *replacing* these characters in Table 1 will be the same set of the characters in that table.

Recall that the sequence of characters obtained from all the texts used and subjected to the treatment described in section 2 is called $S_1$. If each character in $S_1$ is replaced by the corresponding character according to the permutation specified, a new sequence of characters is obtained and it will be called $S_2$. (It is clear, therefore, that the sequence of characters $S_1$ is the entity to which the permutation obtained from the characters in Table 1 is applied.)

A new binary string ($B_2$) may be obtained from $S_2$ by applying the same procedure used to obtain the binary string $B_1$ from $S_1$. This new binary string ($B_2$) also passes (as did $B_1$) the statistical tests specified in 4.2. The corresponding results are presented in section 4.3.

As stated above, $B_2$ may be used with an extremely high degree of security not only in the fields of statistics and computer simulation, but also in that of cryptography. In fact, consider the case of a person P who does not have access to that binary string – in any particular form – or to a system which generates it automatically. To obtain $B_2$, P would have to make correct guesses not only about a) which literary works were used, and b) exactly how this material was processed to obtain $S_1$, but also about what permutation was used to obtain $S_2$ from $S_1$. However, the probability that the specific permutation would be guessed is very low. Actually there are 128! possible permutations of 128 elements and there is no reason to suppose that the probability of choosing any particular one of those would be greater than that of choosing any one of the others. Consequently, it may be accepted that the probability of P making a correct guess about which permutation was actually selected is equal to $\frac{1}{128!}$. In other words, the probability that the guess is wrong is equal to $1 - \frac{1}{128!}$. Thus it is almost certain that P's guess would be incorrect. For this reason, $B_2$ – obtained from $S_2$ – really is useful in the field of cryptography as well [2].

Once one has $B_2$, new permutations of the 128 characters in Table 1 can be generated automatically. Each of these new permutations makes it possible to obtain a new binary string which can be useful in the fields of statistics, computer simulation and cryptography. In this section an explanation is provided about how to use $B_2$ to obtain a permutation of these 128 characters. A line segment is considered as an interval whose initial extreme, or point, corresponds to 0 and whose final extreme, or point, corresponds to 1. In addition, 128 equal-length subintervals are considered in that interval. The characterization of those subintervals is as follows: "subinterval 1" is that whose extreme points are 0 and $\frac{1}{128}$, and "subinterval 2" is that whose extremes are $\frac{1}{128}$ and $\frac{2}{128}$, and so forth until reaching the last of those subintervals, "subinterval 128", whose extremes are $\frac{127}{128}$ and 1. By using $B_2$, a number is generated in the interval ranging from 0 to 1. (With this objective the procedure can be used to generate probabilities from a binary string, as specified in section 1.2.) Suppose, for example, that one is using a precision of 7 decimal digits. If the number thus generated belongs only to "interval 1", character $c_1$ in Table 1 will be considered to be chosen. If the number generated belongs only to "interval 2", character $c_2$ in Table 1 will be considered to be chosen, and so on successively. (Thus, for instance, if the number generated is 0.5196774, which belongs only to "subinterval 67", character $c_{67}$ will be considered to be selected from the table.)

Given the precision under which we are operating, how can one proceed if the number generated corresponds to one of the extremes shared by any two of the adjoining subintervals mentioned above? In that case, one way of proceeding is to discard the number generated and generate a new number. This criterion will continue to be used until the number generated no longer corresponds to any of the extreme points. Suppose that one of the 128 characters in Table 1 was already chosen. Then, every time character $c_1$ appears in $S_1$, it will be replaced by the character selected as indicated above. If, for example, character 67 ($c_{67}$) was selected from Table 1, every time $c_1$ appears in $S_1$, this character will be replaced by $c_{67}$. The remaining 127 characters in the table are renumbered from 1 to 127, thus preserving the increasing order of the subscripts identifying them. (In this case, the first 66 characters of Table 1 will continue to be numbered in the same way, but according to the renumeration mentioned, number 67 will correspond to character $c_{68}$, number 68 will correspond to $c_{69}$, and so on, until reaching character $c_{128}$ of the table, which corresponds to number 127.) Then, 127 subintervals of equal length are considered in that interval having 0 and 1 as extremes. The first of these subintervals, "subinterval 1", will have points 0 and $\frac{1}{127}$ as extremes, "subinterval 2" will have points $\frac{1}{127}$ and $\frac{2}{127}$ as extremes, and so forth. Therefore, "interval 127" will have points $\frac{126}{127}$ and 1 as extremes. A new number with 7 decimal digits between 0 and 1 is generated from $B_2$. If, with the precision used, this new number corresponds to one of the extremes of two of the new subintervals, it is discarded and another number is generated. This criterion continues to be applied until the number generated belongs only to one of the 127 subintervals considered. There will be some character in Table 1 such that the number which it was assigned using the renumbering process described above is the same as the last number generated. That is the second character selected from Table 1. Then, every time that $c_2$ appears in $S_1$, it will be replaced by the second character selected. On each occasion, of course, consideration is given to a number of subintervals within the subinterval extending from 0 to 1, equal to the number of characters of Table 1 which have not yet been chosen. In this way, a permutation of the first 128 characters in Table 1 may be obtained. If this permutation is the same as that which led to the generation of $S_2$ from $S_1$, it is discarded and another new permutation of those characters is obtained. This criterion continues to be applied until obtaining a permutation different from that which led to the generation of $S_2$ from $S_1$. The replacements corresponding to the different characters are made in $S_1$ according to the new permutation. Hence a sequence of characters ($S_3$) is obtained. Using the procedure described in section 2, a new binary string is obtained from $S_3$ and will be called $B_3$. The binary string $B_2$ can be made longer with the binary string $B_3$.

Using $B_2$ repeatedly to obtain new permutations of the characters in Table 1, new binary strings may be obtained to make the existing binary string longer. It is essential to discard any permutation of characters in Table 1 which is the same as another that was obtained previously, so that no binary string will be repeated.

## 4 Generators of binary strings tested for randomness

The physical-mathematical system characterized in section 2, which is an initial approximation to an IRBG, will be referred to here as $(HRNG)_1$. This random-number generator (RNG) can be considered to be hybrid (H) since it is not purely mathematical; that is, it also includes a physical component. The literary texts are the "raw material" from which the random binary string is obtained, thus making it possible to generate, for example, random numbers expressed in base 10 and probability values. Subscript 1 in the expression $(HRNG)_1$ refers to the fact that this generator is the first of its type that was obtained before using the different permutations of the characters in Table 1, as was done to obtain a second approximation to the IRBG.

Here three particular cases of this second approximation, obtained by using different sets of permutations of characters in Table 1 (such that any two of those sets are disjoint; i.e., they do not include, as elements, any permutations in common) will be called $(HRNG)_2$, $(HRNG)_3$ and $(HRNG)_4$. In future work in which this type of random number generator is used with permutations of the characters in Table 1, it will generally be referred to as HRNG.

In addition to $(HRNG)_1$, $(HRNG)_2$, $(HRNG)_3$, and $(HRNG)_4$, two common pseudorandom generators were used to generate binary strings: a) the implementation of a congruential algorithm incorporated into the Java 2 platform (version 1.6), and the implementation of the same algorithm incorporated into the runtime library of Microsoft's Visual C++ 2008 compiler. They were subjected to the randomness tests described in section 5.

Both Java and Visual C++ generate pseudorandom numbers with a code based on a linear congruential algorithm used by Knuth [3], which can be expressed by a single equation:

$$X_{n+1} = (aX_n + c) \mod m \tag{7}$$

$X_0$ is the initial "seed" for the pseudorandom sequence. Given the same "seed", the algorithm will produce the same sequence of pseudorandom numbers as output.

The set of parameters ($a$, $c$ y $m$) used in Java [11] is different from the corresponding set used in Visual C++. (Standard references regarding this selection are not readily available, so the corresponding source code was consulted to obtain this information.)

Both the implementation of the algorithm used in Java and that used in Visual C++ offer the user the option of obtaining as many times as necessary certain binary sequences: one composed of 32 bits in the case of Java and one of 16 bits in the case of Visual C++.

As expressed above, using the Java generator a 32-bit pseudorandom binary string can be generated. This process can be repeated as many times as required. Thus the first 32-bit binary string is generated. Then the second binary string having the same length is generated. This second binary string is considered to be a prolongation or a continuation of the first string. Using the same generator, a third 32-bit binary string is generated. This third 32-bit string is considered to be a continuation of the binary string formed by lengthening the first binary string with the second. The process can be continued until the binary string is as long as desired according to one's objectives.

In addition, by using the Visual C++ generator repeatedly to form 16-bit binary strings, applying the same procedure described above for binary strings produced with the Java generator, a binary string of the desired length can be obtained. Suppose, for example, that one requires a 1853-bit binary string. Using the Visual C++ generator, it is possible to obtain 116 binary strings of 16 bits each. In this way a 1856-bit binary string may be obtained. If the last three (or the first three) bits of this string are discarded, the desired string is obtained.

## 5 Randomness tests applied and the results obtained

The different generators described in section 4 were used to form binary strings which were tested for randomness. These generators include four introduced in this article – $(HRNG)_1$, $(HRNG)_2$, $(HRNG)_3$, and $(HRNG)_4$ – and two others which are highly used (to be called Java and C++ for short).

In this section, a discussion will be provided of 1) the statistical tests applied to the binary strings obtained by using these generators, and 2) the results obtained by using those tests.

Let there be a genuinely random binary string like that generated by the IRBG. The expression $p(0,0)$ will be used to refer to the probability that a dyad (i.e., a sequence of two consecutive bits) selected at random from that string will be (0, 0). Likewise, the expressions $p(0,1)$, $p(1,0)$, $p(1,1)$ will refer to the probabilities that the dyad will be $(0,1)$, $(1,0)$ and $(1,1)$, respectively. For the type of binary string specified, the following should be fulfilled: $p(0,0) = p(0,1) = p(1,0) = p(1,1) = \frac{1}{4}$. In other words, there is no reason to suppose that in a random binary string there would be a tendency for any one of the four possible dyads to be present, different from the tendency of any of the other three dyads to be present.

The expression $p(0,0,0)$ will be used to refer to the probability that a triad (i.e., a sequence of three consecutive bits) randomly chosen from that random binary string will be (0, 0, 0). Likewise, the expressions $p(0,0,1)$, $p(0,1,0)$, $p(0,1,1)$, $p(1,0,0)$, $p(1,0,1)$, $p(1,1,0)$ and $p(1,1,1)$ will refer, respectively, to the probabilities that the triad will be $(0,0,1)$, $(0,1,0)$, $(0,1,1)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$ and $(1,1,1)$. For the type of binary string specified, the following should be fulfilled: $p(0,0,0) = p(0,0,1) = p(0,1,0) = p(0,1,1) = p(1,0,0) = p(1,0,1) = p(1,1,0) = p(1,1,1) = \frac{1}{8}$. In other words, there is no reason to suppose that in a random binary string there would be a tendency for any one of the eight possible triads to be present, different from the tendency of any of the other seven triads to be present.

Clearly, the preceding idea can be generalized for the sixteen possible tetrads, for the thirty-two possible pentads, etc.

In a binary string, given any two of its consecutive bits, there are four possible types of transitions from the first to the second: from a 0 to a 0, from a 0 to a 1, from a 1 to a 0, and from a 1 to a 1. If the given binary string is random, the probability that a certain transition chosen at random will be the same as the probability that it will be any of the other three. In other words, there is no reason to suppose that in a random binary string there would be a tendency for any one of the four possible transitions to be present, different from the tendency of any of the other three transitions.

The term "length of a binary string", $L_s$, will be used to refer to the number of bits comprising it. Given the above considerations, one can calculate, for random binary strings of various lengths, which are, from a theoretical perspective, the most probable numbers for the different dyads, triads, tetrads, pentads and transitions that will be present in them. In addition, one can count how many different dyads, triads, tetrads, pentads and transitions are actually present in those binary strings. (The numerical values counted are usually also known as "values observed".) It follows quite naturally then that the non-parametric statistical chi-square ($\chi^2$) test may be applied to determine whether, with a given level of significance, the differences found between the expected numerical values based on theoretical considerations and the corresponding numerical values actually observed are significant. This test was used, in all cases with a level of significance of 0.05, with the objective indicated above.

Note that, for these cases, the null hypothesis (i.e., the hypothesis that these differences are not significant) is precisely the hypothesis that we wish to prove in this study.

The results of the $\chi^2$ test are shown in Table 2, with a level of significance of 0.05 and the degrees of freedom pertinent for the different cases (3 for transitions and dyads, 7 for the triads, 15 for the tetrads and 31 for the pentads) when applied to 1000 binary strings of specified lengths, formed by each of the different generators used. It can be noted that in all cases the percentages corresponding to the numbers of binary strings such that each of them does not pass the $\chi^2$ test are greater for the Java and C++ generators than for the generators introduced here.

| Transitions | $L_s = 8001$ | | | | | |
|---|---|---|---|---|---|---|
| test | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| Passed test | 961 | 951 | 951 | 950 | 930 | 921 |
| Failed test | 39 | 49 | 49 | 50 | 70 | 79 |
| Failures (%) | 3.9% | 4.9% | 4.9% | 5.0% | 7.0% | 7.9% |

| Dyads | $L_s = 8000$ | | | | | |
|---|---|---|---|---|---|---|
| test | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| Passed test | 957 | 960 | 954 | 950 | 930 | 923 |
| Failed test | 43 | 40 | 46 | 50 | 70 | 77 |
| Failures (%) | 4.3% | 4.0% | 4.6% | 5.0% | 7.0% | 7.7% |

| Triads | $L_s = 16000$ | | | | | |
|---|---|---|---|---|---|---|
| test | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| Passed test | 956 | 961 | 954 | 950 | 925 | 930 |
| Failed test | 44 | 39 | 46 | 50 | 75 | 77 |
| Failures (%) | 4.4% | 3.9% | 4.6% | 5.0% | 7.5% | 7.9% |

| Tetrads | $L_s = 32000$ | | | | | |
|---|---|---|---|---|---|---|
| test | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| Passed test | 957 | 954 | 954 | 953 | 938 | 939 |
| Failed test | 43 | 46 | 46 | 47 | 62 | 61 |
| Failures (%) | 4.3% | 4.6% | 4.6% | 4.7% | 6.2% | 6.1% |

| Pentads | $L_s = 64000$ | | | | | |
|---|---|---|---|---|---|---|
| test | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| Passed test | 974 | 970 | 952 | 955 | 935 | 940 |
| Failed test | 26 | 30 | 49 | 45 | 65 | 60 |
| Failures (%) | 2.6% | 3.0% | 4.9% | 4.5% | 6.5% | 6.0% |

Table 2: $\chi^2$ tests.

Another test to which the binary strings obtained with the generators considered in this paper were subjected is based on the following statistical result: The binomial distribution for which each of the two possible events has the same probability of occurring is an excellent approximation to the normal distribution, for very numerous sets of data. Suppose that by using one of these generators, such as $(HRNG)_1$, 100,000 binary strings are obtained, where $L_s = 1000$ for each. If a graph is made of the number of binary strings which include exactly a particular number of "ones" based on that quantity, a good approximation to a normal curve can be obtained. (Of course, for each of the 100,000 strings the number of "ones" included in it could vary from 1 to 1,000.) See this graph in Figure 1. (The dotted line represents the binomial distribution; and the solid curve, the corresponding normal distribution.) The same procedure was used to present graphs of the same type corresponding to the binary strings generated by $(HRNG)_2$, $(HRNG)_3$, $(HRNG)_4$, Java and C++, in Figure 2, Figure 3, Figure 4, Figure 5 and Figure 6, respectively. Note that in these graphs the approximations to the normal curve corresponding to the binomial distributions obtained by using the generators introduced here are better than the approximations to the normal curve corresponding to the binomial distributions obtained with the Java and C++ generators.

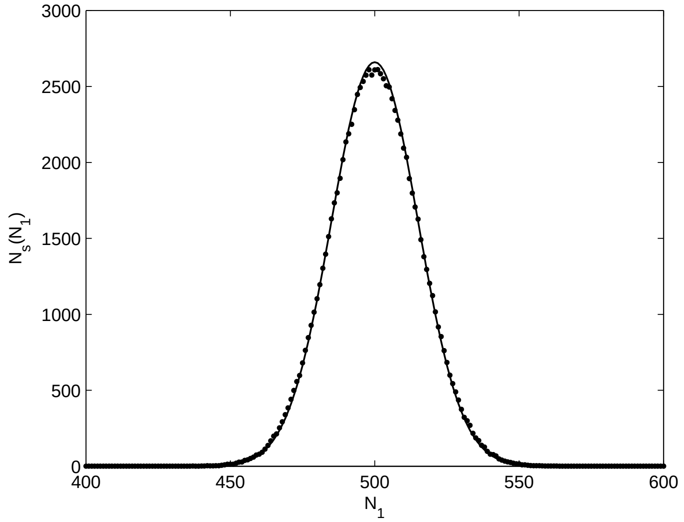

Figure 1: Distribution for 100,000 binary strings generated by $(HRNG)_1$. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

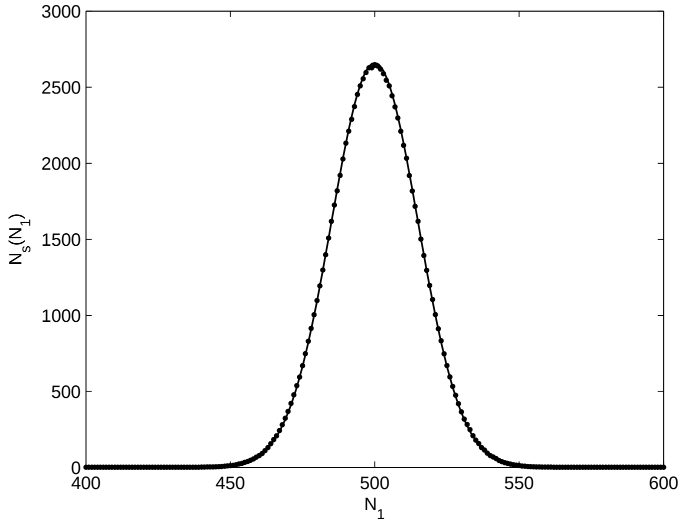

Figure 2: Distribution for 100,000 binary strings generated by $(HRNG)_2$. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

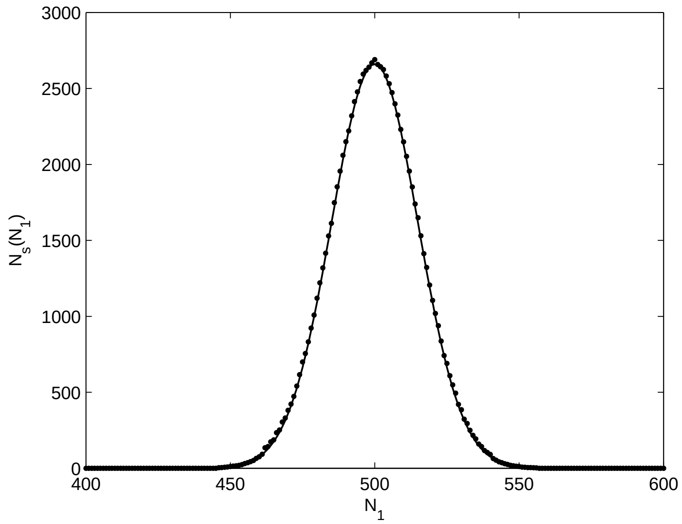

Figure 3: Distribution for 100,000 binary strings generated by $(HRNG)_3$. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

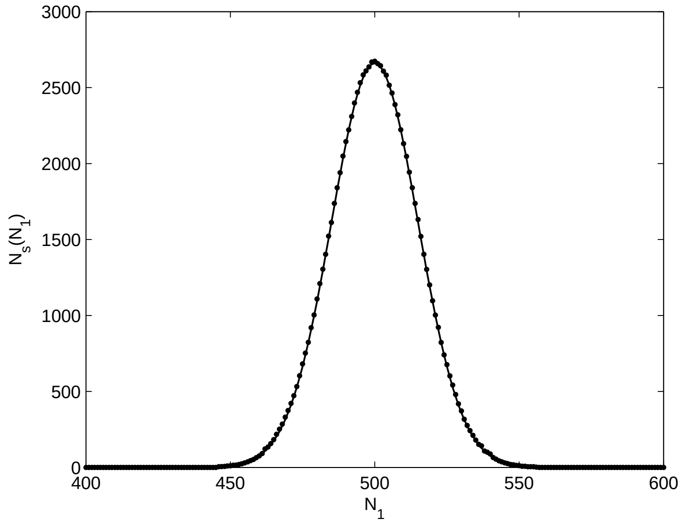

Figure 4: Distribution for 100,000 binary strings generated by $(HRNG)_4$. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

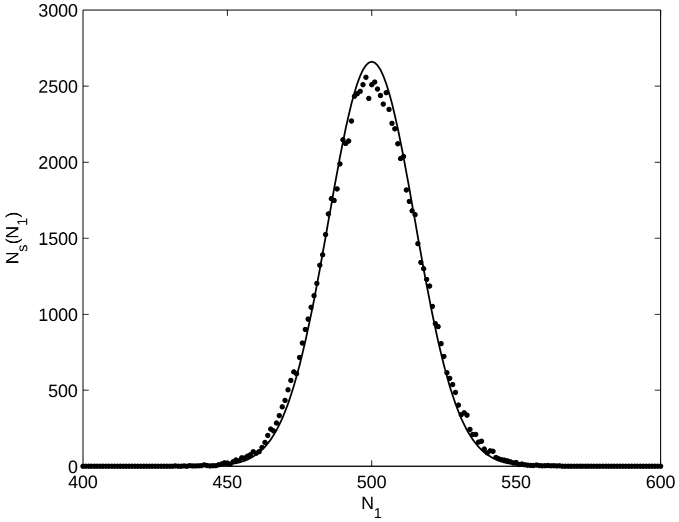

Figure 5: Distribution for 100,000 binary strings generated by Java. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

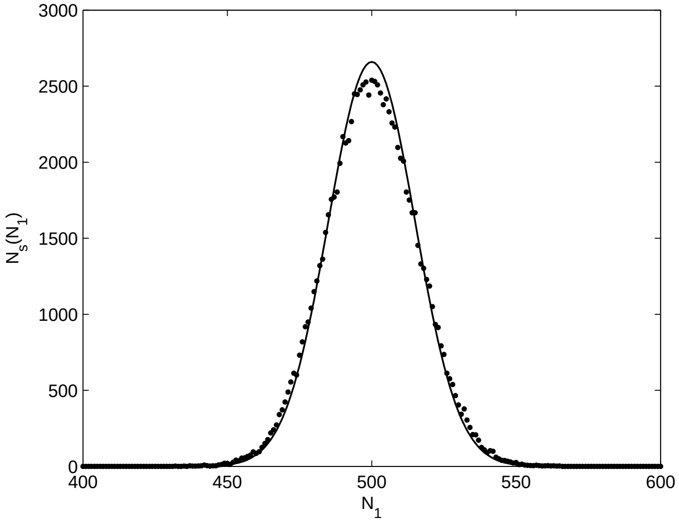

Figure 6: Distribution for 100,000 binary strings generated by C++. $N_1$ refers to the number of 'ones' generated in each string; and $N_s(N_1)$ refers to the number of binary strings as a function of $N_1$.

Another way to appreciate the quality of the approximations (of the binomial distributions considered) to the normal distribution is as follows. Let $\mu$ and $\sigma$ be the mean and standard deviation, respectively, of each of the distributions considered. For the normal curve, the percentage of the area under the curve from point $\mu - \sigma$ to point $\mu + \sigma$ is approximately 68.26%. This value can be seen in Table 3, along with two others.

| Interval | Percentage of area under curve |
|---|---|
| From $\mu - \sigma$ to $\mu + \sigma$ | 68.26% |
| From $\mu - 2\sigma$ to $\mu + 2\sigma$ | 95.44% |
| From $\mu - 3\sigma$ to $\mu + 3\sigma$ | 99.74% |

Table 3: Normal reference curve.

Information of the same type as that displayed in Table 3 is shown in Table 4 for the binomial distributions formed by the generators discussed in this paper.

| Interval | Percentage of area under curve | | | | | |
|---|---|---|---|---|---|---|
| | $(\text{HRNG})_1$ | $(\text{HRNG})_2$ | $(\text{HRNG})_3$ | $(\text{HRNG})_4$ | JAVA | C++ |
| $\mu - \sigma$ to $\mu + \sigma$ | 68.20% | 68.27% | 68.27% | 68.26% | 67.16% | 67.19% |
| $\mu - 2\sigma$ to $\mu + 2\sigma$ | 95.54% | 95.35% | 95.86% | 95.45% | 94.45% | 94.43% |
| $\mu - 3\sigma$ to $\mu + 3\sigma$ | 99.70% | 99.72% | 99.75% | 99.74% | 99.49% | 99.39% |

Table 4: Binomial distributions.

The comparison of data seen in Table 3 and Table 4 makes it possible to note once again that the approximations to the normal curve corresponding to the binomial distributions obtained by using the generators presented here are better than the approximations to the normal curve corresponding to the binomial distributions obtained by using the Java and C++ generators.

The concept of "index of randomness" for $m$-ary strings, where $m = 2, 3, 4, \dots$, was introduced in a previous article [10]. (In particular, this index can be computed for strings such that $m = 2$; that is, for binary strings.) It is feasible to compute the most probable value of the average of the indexes of randomness of a certain number, such as 25,600, of binary strings that could potentially be generated by the IRBG, such that for each of them $L_s = 8$. There are 256 different binary strings with $L_s = 8$. For none of them is there a tendency to be generated by the IRBG which is different from the tendency to be generated for any of the other 255 strings. First, then, the randomness index corresponding to each of the aforementioned binary strings is computed. Second, the average is found of the 256 values of the randomness indexes computed. That average is precisely the most likely value sought. In addition, each of the generators considered here is used to form 25,600 binary strings such that, for each of them, $L_s = 8$. With each of the sets of 25,600 binary strings, the following is done: First, the randomness index is computed for each of the 25,600 binary strings. Second, the average of the 25,600 randomness indexes already computed is found.

The same approach is used to compute the most probable value of the average of the indexes of randomness of a certain number, such as 6,553,600, of binary strings that could potentially be generated by the IRBG, such that for each of them $L_s = 16$. There are 65,536 different binary strings with $L_s = 16$. For none of them does there exist a tendency to be generated by the IRBG which is different from the tendency to be generated for any of the other 65,535 strings. First, then, the randomness index corresponding to each of the 65,536 above-mentioned binary strings is computed. Second, the average is found of the 65,536 values of the randomness indexes computed. That average is precisely the most likely value sought. In addition, each of the generators considered here is used to generate 6,553,600 binary strings such that for each of them $L_s = 16$. With each of the sets of 6,553,600 binary strings, the following is carried out: First the randomness index is computed for each of the 6,553,600 binary strings. Second, the average is calculated of the 6,553,600 randomness indexes already computed.

The results are given in Table 5. It can be seen that according to the values of the randomness indexes computed, the generators described here are better approximations to the IRBG than those of Java and C++.

# 6 Discussion

## 6.1 Support for the main hypothesis of this paper

As indicated in section 2, it is possible to determine the frequencies of appearance of the different ordered pairs of characters in a sequence of those pairs actually used in an HRNG. Thus, for example, seven pairs of

| | $L_s = 8$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | IRBG | $(\mathrm{HRNG})_1$ | $(\mathrm{HRNG})_2$ | $(\mathrm{HRNG})_3$ | $(\mathrm{HRNG})_4$ | JAVA | C++ |
| Avg. | 0.4238 | 0.4234 | 0.4235 | 0.4233 | 0.4241 | 0.4002 | 0.4591 |
| Min. | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Max. | 0.6466 | 0.6464 | 0.6464 | 0.6464 | 0.6466 | 0.6890 | 0.6300 |

| | $L_s = 16$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | IRBG | $(\mathrm{HRNG})_1$ | $(\mathrm{HRNG})_2$ | $(\mathrm{HRNG})_3$ | $(\mathrm{HRNG})_4$ | JAVA | C++ |
| Avg. | 0.5445 | 0.5442 | 0.5452 | 0.5442 | 0.5442 | 0.5454 | 0.5454 |
| Min. | 0 | 0 | 0 | 0 | 0 | 0 | |
| Max. | 0.7113 | 0.7112 | 0.7112 | 0.7112 | 0.7112 | 0.7101 | 0.7154 |

Table 5: Randomness indexes.

these frequencies were determined for an HRNG corresponding to that which was introduced as the second approximation to an IRBG. Each of these pairs corresponded to a frequency of appearance of a certain pair of characters and to the frequency of the pair ordered symmetrically with regard to the prior one. Thus, for example, $f\{a, e\}$ and $f\{e, a\}$ were determined. These frequencies are the best approximations available for the probabilities $p\{a, e\}$ and $p\{e, a\}$, respectively. In general, if $c_i$ and $c_j$ are any two of the characters in Table 1, $f\{c_i, c_j\}$ and $f\{c_j, c_i\}$ would be the best approximations available for $p\{c_i, c_j\}$ and $p\{c_j, c_i\}$, respectively. However, if the main hypothesis of this study is valid (equation (4) in section 2), then $f\{c_i, c_j\}$ and $f\{c_j, c_i\}$ should be, as mentioned in section 2, approximately equal.

$$f\{c_i, c_j\} \simeq f\{c_j, c_i\}. \tag{6}$$

Seven examples of this type are presented below; they have been obtained for a total of $4,429,227,264$ ordered pairs of characters actually used.

$$\begin{aligned}
f\{a, e\} = 0.00581045 &\quad\simeq\quad f\{e, a\} = 0.00581117 \\
f\{d, f\} = 0.0050659 &\quad\simeq\quad f\{f, d\} = 0.0050797 \\
f\{D, t\} = 0.00005488 &\quad\simeq\quad f\{t, D\} = 0.00005494 \\
f\{k, z\} = 0.00000724 &\quad\simeq\quad f\{z, k\} = 0.00000713 \\
f\{l, s\} = 0.00149778 &\quad\simeq\quad f\{s, l\} = 0.00148978
\end{aligned}$$

$$\begin{aligned} f\{m,o\} &= 0.00101311 &\simeq\quad f\{o,m\} &= 0.00101905 \\ f\{n,r\} &= 0.00242873 &\simeq\quad f\{r,n\} &= 0.00242742 \end{aligned}$$

Equally satisfactory results were obtained for other pairs of frequencies of the type $f\{c_i, c_j\}$ and $f\{c_j, c_i\}$. These results are a strong element of support indicating that the main hypothesis of this paper is correct.

### 6.2 Permutations of the characters in Table 1

It can be proven that if the number of permutations mentioned in 3.2 is very small – for example, the same as or less than 100 – compared to the total number of permutations existing of the characters in Table 1, then the probability that any two of them will coincide or will have a high degree of similarity is very small.

Although some potential users of the HRNG may accept that the preceding statement is correct (an issue to be taken up in detail in another paper), they may be concerned about the possibility that those highly unlikely events may actually occur and produce binary strings with undue regularities. (In this paper, for which 43 permutations of this type were used, none of these extremely unlikely situations occurred.) Fortunately, the nature of the HRNG makes it possible, quite naturally, to use diverse procedures to prevent these hypothetical regularities. Only one of them will be addressed below.

It is important to remember that "short leaps" of 50 characters and "long leaps" of 1600 characters were used. A reason was given for the choice of a minimum for the "short leap", but no reason was given for having chosen precisely 1600 characters as the "length" of each "long leap". Actually, if for the reasons specified in section 2, a "short leap" of 50 characters provides a very high degree of security regarding the independence of the "target" character with respect to the "source" character, it can be accepted that any "long leap" which is at least 5 times longer than the "short leap" of 50 characters is enough to be completely sure about that type of independence: that of the "target" character with respect to the "source" character. It is possible, therefore, to fix the value the "short leap" at 50 characters but use the "long leap" as a parameter to which different amounts of characters may be assigned, such as from 501 to 3001 characters, in the following way: $501, 551, 601, \ldots, 3001$. In other words, the difference between any two consecutive "long leaps" from the above list is 50 characters. Thus, 51 different types of "long leaps" are available. Suppose now that it has been decided to use 51 different permutations to produce a certain set of HRNG-type generators, such as $((HRNG_2)$, $(HRNG)_3$, $(HRNG)_4)$. (Recall that to construct each one of these generators, a different subset – of the set of those 51 permutations – should be used, such that any two of those subsets are disjoint.) Then, for the first permutation a "long leap" of 501 characters will be used; for the second, 551 characters, and so on successively. In this way, for the last, a "long leap" of 3001 characters will be utilized. The simultaneous use of these two techniques – the use of (a) different permutations of characters, and (b) different lengths of "long leap" – is a way to ensure that none of the binary strings generated will have any undue regularities.

Aside from the above, it is important to remember the criterion generally used to evaluate the quality of any pseudorandom or random number generator: the quality of the generator is considered acceptable if the products it generates pass certain randomness tests, and unacceptable if it does not pass them. The products of the HRNG have passed the tests described in section 5. To evaluate to the quality of the HRNG, it has not been deemed reasonable to apply a criterion different from that used to evaluate the quality of any other random or pseudorandom number generator.

# 7 Conclusions and prospects

## 7.1 Conclusions

The physical mathematical system developed makes it possible to generate random numbers with the required amount of digits, expressed in base 10. It also permits the generation of random probabilities, as shown above, with the desired amount of decimal digits.

Four important advantages of the generators $(HRNG)_2$, $(HRNG)_3$ $(HRNG)_4$, etc. include the following:

1. The synthesis of those generators does not require special physical or electronic resources. For this process, a digital computer and adequate software based on the procedure discussed above will suffice.

2. These generators are adequate in the field of statistics, in that of the computational simulation of systems of probabilistic behavior and in that of cryptography.

3. The amount of literary works available in digital format is enormous. Therefore, the possibility of this approach "using up" the resources from which to generate random numbers cannot be a limiting factor.

4. If an infinite sequence of numbers were produced by a generator of the type described here, it would *not* be a periodic sequence, like an infinite numerical sequence would be if it were produced by a congruential generator, such as that used in Java and C++.

   Suppose that an infinite sequence of numbers were produced by a congruential pseudorandom number generator. This sequence would necessarily be comprised of infinite repetitions of one of its subsequences. (In other words, it would be an "infinite periodic numerical sequence".)

One disadvantage of the HRNG presented is the following: Its use demands a much larger amount of memory than that required by common pseudorandom number generators. In effect, the "product" of the HRNG to be used should be a binary string made up of 1,000,000,000 bits. This sequence of bits requires the corresponding amount of computer memory. (Considering the current state of computer technology, this memory is not terribly demanding for a microcomputer.) Pseudorandom number generators do not require this amount of computer memory because they generate their "products" in "real time", whenever the numbers are needed.

## 7.2 Prospects

Any of the generators introduced here for which a permutation of the characters in Table 1 was used in the construction process can be applied in the field of cryptography. Although this topic will be covered in another paper, the essence of what makes this possible will be summarized briefly below.

Let there be a message composed of a sequence of characters. Clearly each character can be encrypted with a sequence of bits. Thus, the message may finally be expressed with a series of bits (i.e., by a binary string $M$). Suppose that the length of that string is 50,000 ($L_s = 50,000$). Then with one of the generators introduced here, for which a permutation of the characters in Table 1 was used in the construction process (that is, one of the generators $(HRNG)_2$, $(HRNG)_3$, or $(HRNG)_4$, etc.), another binary string $G$ whose length is also equal to 50,000 is generated. How can yet another binary string (also with $L_s = 50,000$), which is an encrypted version of $M$, be obtained from these binary strings? As follows: Observe the first bits of both binary strings $M$ and $G$. If these bits are the same (i.e., if both are "zeroes" or if both are "ones"), a "one" is assigned to the first bit of the encrypted message $C$. On the contrary, if both bits are different, a "zero" is assigned to that first bit. Then the second bits of $M$ and $G$ will be considered. If both bits are the same, a "one" will be assigned to the second bit in $C$. If, on the contrary, they are different, a "zero" will be assigned to that second bit in $C$. This process will continue until the last bit of the encrypted message has been obtained. Figure 7 illustrates this.

```
M: 1 0 0 1 0 0 1 1 1 0 1 ... 0 1 0 0 1 0
G: 0 1 0 0 0 1 1 1 0 1 1 ... 1 0 0 1 1 0
C: 0 0 1 0 1 0 1 1 0 0 1 ... 0 0 1 0 1 1
```

Figure 7: One way of obtaining $C$, an encrypted version of $M$

It is evident that a receiver of an encrypted message $C$, who has $G$ or the possibility of generating $G$, can decipher $C$ (i.e., regenerate $M$).

This study is a product of an ongoing research project "Analysis of fluctuations" of the Universidad Nacional (Costa Rica). It is being conducted by members of the Applied Mathematics and Computer Simulation Group and has been oriented toward developing a resource which will be used as a tool in a project requiring a great deal of tasks related to the simulation of probabilistic systems which must be carried out with no undesirable bias.

A future paper will be devoted to another generator, a purely mathematical generator, which will be able to "compete" with any of the generators $(HRNG)_2$, $(HRNG)_3$, etc., introduced here.

## References

[1] Chaitin, G. J. (2001) *Exploring Randomness.* Springer, Berlin.

[2] Kelsey, J.; Schneier, B.; Wagner, D.; Hall, C. (1998) "Cryptanalytic attacks on pseudorandom number Generators", in: *Fast Software Encryption, Fifth International Workshop Proceedings (March 1998).* Springer, Berlin: 168–188.

[3] Knuth, D. (1998) *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Addison-Wesley, Reading, MA.

[4] Li, M.; Vitanyi, P. (1997) *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, Berlin.

[5] Marsaglia, G.; Tsang, W. (2002) "Some difficult-to-pass tests of randomness", *Journal of Statistical Software* **7**(3).

[6] Menezes, A; van Oorschot, P.; Vanstone, S. (1996) *Handbook of Applied Cryptography*. CRC Press, New York.

[7] National Institute of Standards and Technology (2002) *A Statistical Test Suite for Random and Pseudorandom Number Generators*, Gaithersburg, Maryland.

[8] Pincus, S.; Singer, B. H. (1996) "Randomness and degrees of irregularity", *Proceedings of the National Academy of Sciences of the United States of America* **93**: 2083–2088.

[9] RAND Corporation (2002) *A Million Random Digits with 100,000 Normal Deviates*. American Book Publishers, Salt Lake City, Utah.

[10] Skliar, O.; Monge, R. E.; Oviedo, G.; Medina, V. (2009) "Indices of regularity and indices of randomness for $m$-ary strings", *Revista de Matemática: Teoría y Aplicaciones* **16**(1): 43–59.

[11] Sun Microsystems (2003) "Random", *Java 2 Platform, Standard Edition*, URL:
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html.
Accessed I-03-2010.

[12] Volchan, S. B. (2002) "What is a random sequence?" *American Mathematical Monthly* **109**(1): 46–68.

[13] von Neumann, J. (1951) "Various techniques used in connection with random digits", in: A. S. Householder, G. E. Forsythe & H. H. Germond (Eds.) *Monte Carlo Method*, National Bureau of Standards Applied Mathematics Series. Government Printing Office, Washington, D.C.: 36–38.