Randomness of Finite Strings: A Reconstructive Approach

Sami Khuri
(408)924-5081
khuri@sjsumcs.sjsu.edu

Frederick Stern
(408)924-5155
stern@sjsumcs.sjsu.edu

Teresa Chiu
(408)255-0279
chiu@jupiter.sjsu.edu

Department of Mathematics and Computer Science
San José State University, One Washington Square
San José, CA 95192-0103

Keywords: *approximation algorithm, finite strings, randomness, reconstruction problem, statistical analysis*

Abstract

We introduce a new concept of randomness for binary strings of finite length, *reconstructive randomness*. Reconstructive randomness is based on a reconstruction problem: putting together subsequences of length k, $k < n$, of a string of length n to uniquely reconstruct it. We claim that the longer the subsequences required to uniquely reconstruct a string of length n, the more random that string is when compared to the other binary strings of the same length. We have statistical evidence supporting this claim. The reconstruction problem approximately translates into a question of whether a system of binary linear equations has zero, one, or many solutions. We develop a method, short of complete enumeration, to solve the latter problem.

1 Introduction

Let $b_1 b_2 ... b_i ... b_n$, where $b_i = 0$ or 1 for $i = 1, 2, ..., n$ be an n-string. We develop a definition of relative reconstructive randomness for n-strings. The definition is based on the k-signature sets of an n-string for each positive integer $k < n$. To illustrate, 4-signature set of the 5-string 00110, denoted by $L_4(00110)$ is the set $\{(0010; 2), (0011; 1), (0110; 2)\}$ where the first component of each pair is obtained by removing one bit at a time from 00110 and concatenating (if necessary) the remaining four bits. Thus, by removing b_1 from 00110 we obtain 0110. The second component in $(0010; 2)$ represents the frequency with which this process yields 0010. Since 0010 is also obtained by deleting b_3 from 00110, the second component of $(0010; 2)$ is two.

Given the set $\{(0010; 2), (0011; 1), (0110; 2)\}$, one can uniquely reconstruct the 5-string 00110 since the latter is the only binary string of length 5 with the given 4-signature set. (We emphasize: in all that follows "reconstruction" refers to this unambiguous situation where a signature set belongs to one and only one string). It can be shown that any 5-string can be uniquely reconstructed from its 4-signature sets. The situation is quite different when 2-signature sets are considered instead of 4-signature sets. Some 5-strings can be reconstructed while others cannot. The former are less random than the latter, according to our definition of reconstructive randomness.

In the next section, we formalize the above observations, and give definitions for reconstructive randomness, and
reconstructability–level of strings. To support our claim that the class with the highest reconstructability level is a good source of random strings (in the standard sense), we investigate the randomness of strings from different reconstructability levels in Section 3. The details of the tests we use are found in the Appendix. Although we believe that there aren't closely related problems to ours in the literature, we mention in Section 4 some problems that tackle similar subjects, but whose objectives are very different than ours. In Section 5, we introduce Algorithm Construct which gives approximate solutions to the problem stated in Section 2.

2 Definition of Reconstructive Randomness

There are $\binom{n}{k}$ ways of removing $n-k$ bits from an n-string, and concatenating the remaining digits. There are at most 2^k distinct k-strings which could be thus formed. The k-signature set of a given n-string records which k-strings occurred, and how frequently they occurred, summarizing the results of $\binom{n}{k}$ different $n-k$ digit removals. The general representation of the **k-signature set of an n-string** $b_1 b_2 ... b_i ... b_n$ is given by:

$$L_k(b_1 b_2 ... b_i ... b_n) = \{(b_{11} b_{12} ... b_{1k}; f_1), (b_{21} b_{22} ... b_{2k}; f_2), ..., (b_{m1} b_{m2} ... b_{mk}; f_m)\}$$

where $m \leq 2^k$, $b_{ij} = 0$ or 1, and $f_i \geq 1$. Here f_i denotes how many times the k-string $b_{i1} b_{i2} ... b_{ik}$ occurred after removing $n-k$ bits and, if necessary, concatenating the remaining digits.

The k-signature set of an n-string $b_1 b_2 b_3 ... b_n$:

$$L_k(b_1 b_2 ... b_i ... b_n) = \{(b_{11} b_{12} ... b_{1k}; f_1), (b_{21} b_{22} ... b_{2k}; f_2), ..., (b_{m1} b_{m2} ... b_{mk}; f_m)\}$$

may come from only one of the 2^n n-strings or it may show up from two or more distinct n-strings. We say that an n-string can be reconstructed from its k-signature set if no other n-string produces the same k-signature set. Whether or not an n-string is reconstructable from its k-signature set

r–level	n=3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	6	12	18	24	30	36	42	48	54	60	66	72	78	84
3	0	2	12	38	92	200	398	754	1324	2228	3496	5240	7426	10128
4	0	0	0	0	4	18	70	220	668	1774	4500	10566	23742	50710
5	0	0	0	0	0	0	0	0	0	32	128	504	1520	4580
6	0	0	0	0	0	0	0	0	0	0	0	0	0	32

Table 1: Approximate number of n-strings at each reconstructability-level ($n = 3$ to $n = 16$).

can be determined quickly for n small enough by complete enumeration: Calculate the k-signature set for each of the 2^n n-strings and determine whether or not the given k-signature set appears exactly once. The following example illustrates that knowing the 2-signature set of 000110 is insufficient for reconstructing it.

Example 1

$L_2(000110) = \{(00;6), (01;6), (10;2), (11;1)\}$. But 001001 also has the same 2-signature set. Since $L_2(001001) = L_2(000110)$, 000110 cannot be reconstructed (uniquely, as our definition requires) from the 2-signature set. □

The above example illustrates two 6-strings that cannot be reconstructed from their 2-signature sets. On the other hand, it can be shown that the 6-string 110100 can be reconstructed from its 2-signature set: $\{(00;3), (01;1), (10;8), (11;3)\}$. It can also be shown that since 110100 can be reconstructed from its 2-signature set, it is reconstructable from its k-signature sets for $k > 2$. It cannot be reconstructed from its 1-signature set: $\{(0;3), (1;3)\}$ since any 6-string with 3 ones and 3 zeros will have the same 1-signature set. We are now ready to put a measure on an n-string based on how difficult its reconstruction from its various k-signature sets is.

The **reconstructability–level** of an n–string $b_1...b_i...b_n$, has value d, denoted by r–level($b_1...b_i...b_n$) $= d$, if it can be reconstructed from its d-signature set but not from its (d−1)-signature set. Note that $d > 1$ except for the n-string 11...1 or 00...0, in which case $d = 1$.

Since it can be shown that each (2n+1)-string has a distinct (n+1)-signature set, it follows that d ≤ n+1 for (2n+1)-strings. Likewise since for $n > 2$, each 2n-string has a distinct n-signature set, $d \leq n$ for 2n-strings. Since each n-string has a unique reconstructability-level, this concept enables a partition of the set of n-strings into equivalence classes of n-strings denoted by their respective reconstructability–levels.

By a method described in Section 5 (Algorithm Construct), we produced Table 1: Approximate number of n–strings at each reconstructability–level ($n = 3$ to $n = 16$). Rather than give exactly the number of n–strings at each r–level, Algorithm Construct supplies an upper bound for these values. The entries of the table give the number of sequences of length n (column) and r–level (row). For instance, there are at most 70 sequences of length 9 whose r–level is 4.

We now define the reconstructive-randomness of an n–string to be its reconstructability level.

Consider two n-strings and their reconstructability levels: r-level(n-string$_i$) $= d_i$ and r-level(n-string$_j$) $= d_j$. We say that n-string$_i$ is more **reconstructive-random** (is more **r–random**) than n-string$_j$, if $d_i > d_j$. In other words, the length of the strings required to reconstruct string$_i$ is greater than the length of the strings needed for the reconstruction of string$_j$. Thus, for example, 001001 is more *r–random* than 110100, since r-level(110100) = 2 and r-level(001001) = 3. If $d_i = d_j$, we say the two strings are equally *r–random*.

We claim that the class of n-strings with the highest r–level (the most reconstructive-random strings) is a potentially good source of random (in the standard sense) strings.

In support of this claim, in the next section, we investigate the randomness of strings from different r-levels.

3 Statistical Analysis

To check our claim that strings with the highest r-level are random in the usual sense, we performed for $n = 15$ some basic statistical tests for randomness [3, 7], that we adapted for the binary case, on four types of strings. All our tests were performed using strings produced by Algorithm Construct. To be able to compare strings from different r-levels, we chose 200 strings and concatenated them from r-level 5 (which is the highest r-level, as can be seen in Table 1). We repeat the process with strings from r-level 4 (there are 23742 strings to choose from), and then once again, we randomly chose 200 strings among the 7426 from r-level 3, and concatenated them together to form another group of strings of length 3000. To use as a control we also randomly generated a string of length 3000 denoted by cp-gen in Table 2. The results of the 8 tests we performed are tabulated in Table 2.

Tests	*3–lev*	*4–lev*	*5–lev*	*cp–gen*
number of ones	50	100	100	100
words of length 2	0	40	100	100
words of length 3	0	40	100	100
words of length 4	0	80	100	100
words of length 5	0	20	100	100
runs of length 2	90	70	100	100
runs of length 3	100	50	100	100
runs of length 4	60	70	100	100
overall results	30	47	100	100

Table 2: Percentage of strings of length 3000 that passed the tests.

We performed two types of tests: the first type tests the occurrence of all possible words in non-overlapping blocks of fixed length. The second type tests runs of ones. The first 5 tests in the table are of the first type, while the last 3 are of the second. The first test simply counts the number of ones in the string. The string passes this test if this count falls in a certain interval of values. We relegate the details of all the tests to the Appendix. The next 4 tests are Chi-square tests which count the number of occurrences of non-overlapping words of length $2, 3, 4$, or 5. The last three tests count the number of runs of ones. The string passes this test if the number of runs lies within a certain interval (see the Appendix for details).

Each test was performed 10 times for each source, and the percentage of passes for each source is recorded in Table 2. We see that only 30% of the strings chosen by concatenating strings from the 3-level passed the test, and only 47% from the 4-level passed the test, while 100% of the strings obtained from the highest r-level passed the tests and as expected, all of the 80 tests performed were passed using the computer randomly generated strings. This result supports our contention that the highest r-level strings are random in the usual sense whereas lower r-level strings are not.

Although our literature search did not produce problems that are very closely related to ours, we mention in the next section the few that tackled problems that are similar, in one way or another, to ours.

4 Related Problems

4.1 Common Subsequences

The following problem studied by Maier, [6] is slightly related to ours. $S = s_1 s_2 ... s_m$ is a sequence with m elements; $|S| = m$. S' is a subsequence of S, $S' < S$, if it is a sequence which consists of S with a number of terms between 0 and m deleted. $R = \{S_1, ..., S_p\}$ is a set of sequences with alphabet A, i.e., the set of values the different $s_i's$ can take.

LCS(R), the longest common subsequence of R, is the longest sequence S, such that $S < S_i$ for i=1,2,...,p.

Similarly, SCS(R), the shortest common sequence of R, is the shortest sequence S', such that $S_i < S'$ for i=1,2,...,p.

Example 2

If R={ababe, cabe, abdec} then

(a) A = {a,b,c,d,e}, and $|A| = 5$,

(b) LCS(R) = abe, and

(c) SCS(R) = cabeabdec

We note that for some problem instances, we might have more than one LCS(R). Nevertheless, they will all have the same length. Similarly for SCS(R).

In our problem, A={0,1} and the k-signature set is our set of sequences. The k-signature set, unlike R, contains the frequency of each subsequence.

4.2 Lempel-Ziv Coding for Data Compression

Considered as the standard for data compression, Lempel-Ziv's algorithm [5] can be thought of as a reconstruction problem. As a matter of fact, data compression's second phase, in which we are to recover the compressed data by decompressing it, is a reconstruction problem.

In the compression phase, the given string, which we denote by $\vec{b} = b_1 b_2 ... b_n$, is parsed into $c(n)$ substrings, called phrases. Each phrase is then coded by a pair to form a set of pairs, $LZ(\vec{b})$. The decompression phase consists in correctly decoding $LZ(\vec{b})$ back to $\vec{b} = b_1 b_2 ... b_n$.

The main difference between this problem and ours is the nature of the $LZ(\vec{b})$ set. While all "phrases" in our k-signature set are of length k, $LZ(\vec{b})$ contains all distinct phrases of $\vec{b}$. We also note that while

$$|LZ(\vec{b})| = c(n) \leq \frac{n}{(1 - \varepsilon_n) \log n}$$

with $\varepsilon_n \to 0$ as $n \to \infty$ [1], we have $|L_k(b_1 b_2 ... b_n)| \leq 2^k$.

4.3 Conditional Kolmogorov Complexity

The conditional Kolmogorov complexity [4], $K_U(\vec{b})$, of a string $\vec{b}$ of length $l(\vec{b})$, with respect to a universal computer U is defined as the minimum length over all programs that output $\vec{b}$ and halt. In other words, by using Cover et al.'s [1] notation, we have

$$K_U(\vec{b} | l(\vec{b})) = \min_{p : \mathrm{U}(\mathrm{p}, l(\vec{b})) \vec{b}} \mathrm{l(p)}.$$

Thus $K_U(\vec{b})$ is the shortest description length of $\vec{b}$ given $l(\vec{b})$, over all descriptions interpreted by computer U. Here lies a slight similarity between Kolmogorov's notion of randomness and *r–random*. The larger the shortest description length $K_U(\vec{b})$, the more random is $\vec{b}$. Analogously, the larger $r\text{–}level(\vec{b})$, the more *r–random* is $\vec{b}$.

In the next section we introduce Algorithm Construct which we use to find the r–level of an n-string.

5 Algorithm Construct

Before formulating the algorithm, we first define the k-sum vector of an n-string, and then derive a few results we use in the algorithm.

The **k-sum vector of** $b_1 b_2 ... b_i ... b_n$ denoted by $S_k(b_1 b_2 ... b_i ... b_n)$, and obtained from the k-signature set $L_k(b_1 b_2 ... b_i ... b_n)$, is a vector of non-negative integers $(s_1, s_2, ..., s_j, ..., s_k)$ where $s_j = \sum_{i=1}^{m} f_i b_{ij}$. That is:

$$S_k(b_1 b_2 ... b_i ... b_n) = \\ (\sum_{i=1}^{m} f_i b_{i1}, \sum_{i=1}^{m} f_i b_{i2}, ..., \sum_{i=1}^{m} f_i b_{iq}, ..., \sum_{i=1}^{m} f_i b_{ik}).$$

In other words, the q^{th} component of $S_k(b_1 b_2 ... b_q ... b_n)$ gives the total number of ones appearing in the q^{th} position of the k-strings in $L_k(b_1 b_2 ... b_q ... b_n)$.

Consider the construction of $b_1 b_2 ... b_6$ from its 3-signature set. There are $\binom{6}{3} = 20$ 3-strings in the set:

$$b_1b_2b_3, b_1b_2b_4, b_1b_2b_5, b_1b_2b_6, b_1b_3b_4, b_1b_3b_5, b_1b_3b_6, \\ b_1b_4b_5, b_1b_4b_6, b_1b_5b_6,$$

$$b_2b_3b_4, b_2b_3b_5, b_2b_3b_6, b_2b_4b_5, b_2b_4b_6, b_2b_5b_6,$$

$$b_3b_4b_5, b_3b_4b_6, b_3b_5b_6,$$

$$b_4b_5b_6.$$

Many of the above 3-strings are identical since there are 2^3 possible 3-strings. We observe that b_1 is the first bit in ten 3-strings, b_2 is the first bit in six 3-strings, b_3 is the first bit in three 3-strings, while b_4 is the prefix of $b_4 b_5 b_6$ only. Thus, the first component of the 3-sum vector of $b_1 b_2 ... b_6$ can be recast in terms of b_1, b_2, b_3, and b_4 as:

$$s_1 = 10 b_1 + 6 b_2 + 3 b_3 + b_4$$

In a similar fashion, s_2 and s_3 can be written as:

$$s_2 = 4b_2 + 6b_3 + 6b_4 + 4b_5$$

$$s_3 = b_3 + 3b_4 + 6b_5 + 10b_6.$$

Given the 3-sum vector of any 6-string, these equations have that 6-string as their unique solution. It follows that any 6-string can be reconstructed from its 3-signature set.

Next we consider the question of whether any 6-string can be reconstructed from its 2-signature set. We are thus given $\{(b_{11}b_{12}; f_1), ...(b_{m1}b_{m2}; f_m)\}$ where $m \leq 4$, from which we can obtain the 2-sum vector (s_1, s_2). As was done with the 3-sum vector, the components of the 2-sum vector can be written as a linear combination of b_1, b_2, b_3, b_4, b_5, and b_6. More precisely:

$$s_1 = 5b_1 + 4b_2 + 3b_3 + 2b_4 + b_5 \quad (1)$$
$$s_2 = b_2 + 2b_3 + 3b_4 + 4b_5 + 5b_6 \quad (2)$$

Example 3

Consider the 6-strings 000110 and 001001 of Example 2. Both strings solve Equations 1 and 2:

$$3 = 5b_1+4b_2+3b_3+2b_4+b_5, \text{ and } 7 = b_2+2b_3+3b_4+4b_5+5b_6.$$

So, neither one can be reconstructed (uniquely, as our definition requires) from its 2-signature set. □

As expected, Algorithm Construct is a generalization of the above procedure. We make use of the k-sum vector and recast its components as a linear combination of the different components of $b_1, b_2, ..., b_n$.

Prior to spelling out our algorithm, we make the following observations. Once the k-sum vector is obtained form the given signature set, the problem is recast as solving a system of linear equations. More precisely, if: $\vec{s}$ is a k×1 column matrix whose components are the k-sum vector's components, and $\vec{b}$ is an nx1 column matrix whose components are $b_1, b_2, ..., b_n$ are to be computed, then finding $\vec{b}$ is equivalent to solving the system: $\vec{s} = A\vec{b}$, where A is a k×n matrix for which:

$$a_{ij} = \begin{cases} \binom{j-1}{i-1}\binom{n-j}{k-i} & \text{for} 1 \leq i \leq k \text{ and } i \leq j \leq n-k+i \\ 0 & \text{otherwise} \end{cases}$$

Note that a_{ij} represents the total number of ways in which $b_j, 1 \leq j \leq n$, can be in the i^{th} position, $1 \leq i \leq k$ of the k-string. While the first $i-1$ bits of the k-string are chosen among the first $j-1$ positions, (i.e., $\binom{j-1}{i-1}$), the remaining $k-i$ bits of the k-string are chosen among the remaining $n-j$ positions, (i.e., $\binom{n-j}{k-i}$).

One more observation about the nature of A will substantially improve the efficiency of our algorithm. The sum of the entries of the columns of A are equal. More precisely,

$$\sum_{i=1}^{k} a_{ij} = \binom{n-1}{k-1} \text{ for } 1 \leq j \leq n$$

which follows from the well-known identity:

$$\sum_{i=0}^{m} \binom{x}{i}\binom{y}{m-i} = \binom{x+y}{m} \text{ for positive integer m.}$$

Thus, by adding the n equations together, we obtain:

$$\sum_{i=1}^{k} s_i = \binom{n-1}{k-1} \sum_{j=1}^{n} b_j$$

In other words, one has to check $\binom{n}{t}$ cases, where

$$t = \sum_{i=1}^{k} \frac{s_i}{\binom{n-1}{k-1}}$$

instead of 2^n vectors.

Algorithm Construct:

Step 1. Compute the k-sum vector from the given signature set:

$$\begin{aligned} &S_k(b_1b_2...b_i...b_n) \\ &= (\sum_{i=1}^{m} f_ib_{i1}, \sum_{i=1}^{m} f_ib_{i2}, ..., \sum_{i=1}^{m} f_ib_{iq}, ..., \sum_{i=1}^{m} f_ib_{ik}) \\ &= (s_1, s_2, ..., s_i, ..., s_n). \end{aligned}$$

Step 2. Compute

$$t = \sum_{i=1}^{k} \frac{s_i}{\binom{n-1}{k-1}}$$

where $\vec{s} = (s_1, s_2, ..., s_i, ..., s_n)$.

Step 3. Solve the system: $\vec{s} = A\vec{b}$, for the $\binom{n}{t}$ strings where $\vec{b}$ has exactly t ones.

- If there is only one solution, then $\vec{b}$ is reconstructable at that level.
- If there is more than one solution, the given $\vec{b}$ is classified as not reconstructable at that level.

Since equality of two k-sum vectors is a necessary but not sufficient condition for equality of the corresponding two k-signature sets, this algorithm gives an upper bound on the number of n-strings in each r-level.

6 Conclusion

We have defined a new concept of relative randomness for binary strings of finite length. We give an algorithm shorter than complete enumeration that yields approximate solutions. Statistical analysis of the most random strings in our sense indicates that they are also random according to classical criteria. Our study of the literature has failed to reveal any closely related approach.

Acknowledgments

The authors would like to thank Thomas Erlebach for comments and suggestions that improved this work. They would also like to thank Stefan Bischof and Helko Lehmann for getting the paper in camera–ready form.

References

[1] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* John Wiley, New York, NY, 1991.

[2] W. Feller. *An Introduction to Probability Theory and its Applications.* Volume I, third edition, John Wiley, New York, NY, 1968.

[3] D.E. Knuth *The art of computer programming: seminumerical algorithms.* Second edition. Addison-Wesley, Reading, Massachusetts, 1981.

[4] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *International Journal of Computer Mathematics*, (2):157–168, 1968. Originally published in Problemy Peredachi Informatsii, 1 (l965).

[5] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22:75–81, 1976.

[6] David Maier. The Complexity of Some Problems on Subsequences and Supersequences. *Journal of the ACM*, 25(2):322–336, April 1978.

[7] R.Y. Rubinstein. *Simulation and the Monte Carlo method.* John Wiley and Sons, New York, 1981.

Appendix: Tests

A) Number-of-Ones Test

Let c be the number of ones in a string of length n. If $c \in (\frac{n}{2} - \frac{3}{2}\sqrt{n}, \frac{n}{2} + \frac{3}{2}\sqrt{n})$, then the string passes this test. The end-points of the interval are for the binomial distribution, the mean minus (or plus) three standard deviations. By the normal approximation to the binomial, the level of significance is about 0.3%. For $n = 3000$, the "passing" interval is $(1441, 1551)$.

B) Words-of-Length-k Test

Assuming that k divides n, a string of length n can be divided into n/k non-overlapping strings of length k. Since there are 2^k different k-length strings, each of these non-overlapping strings of length k belong to exactly one of these categories. For example, if $k = 2$, we can label these categories $00, 01, 10, 11$. Put the categories in lexicographical order and number them from $i = 0$ to $i = 2^k - 1$.

For a given string of length n, let c_i be the number of non-overlapping strings of length k that fall in category i, that is strings that are identical to the integer i in binary notation. The test statistic is

$$q_k = \frac{\sum_{i=0}^{2^k-1}(c_i - \frac{n}{(k2^k)})^2}{\frac{n}{k2^k}}$$

where $\frac{n}{k2^k}$ is the expected number of strings in each of the 2^k categories. Since the test statistic has approximately a Chi-square distribution with $2^k - 1$ degrees of freedom, we use the 99% points of these Chi-square distributions as critical points in our test at a 1% level of significance.

For $k = 2$ (respectively 3, 4, 5), if q_k is less than 11.3 (respectively, 18.5, 30.6, 51) the string passes this test.

C) Runs of Length-r Test

In a string, the first run of r ones occurs where, for the first time, reading from the left to right, a string of r ones appears. For example in 110111011110111 the first run of 3 ones occurs at the end of 110111. To observe the next run, repeat looking for the first run but in a string that begins where the previously observed run ends - in this example 011110111. Thus for the example we have runs of 3 ones occurring where an $*$ is inserted: $110111 * 0111 * 10111*$. In the theory [2], the number of runs of r ones in a string of length n, $R(n;r)$, has, when properly scaled, been shown to have, asymptotically, a normal distribution. In particular,

$$\frac{R(n;r) - n/\mu}{\frac{\sigma}{\mu}\sqrt{n/\mu}}$$

has, asymptotically, a standard normal distribution, where the constants μ and σ^2 are respectively the mean and variance of the reoccurrence times of r-length runs of ones. Here $\mu = 2(2^r - 1)$ and $\sigma^2 = 2^{r+1}[2^{r+1} - (2r+1) - 2^{-r}]$. The "passing" interval for the count of the number of r-length runs in an n-string is determined by

$$(n/\mu - 3\frac{\sigma}{\mu}\sqrt{n/\mu}, n/\mu + 3\frac{\sigma}{\mu}\sqrt{n/\mu}).$$

The significance level is thus about 0.3%. For $r = 2, 3, 4$ with $n = 3000$, the passing intervals are respectively $(449, 551)$, $(176, 252)$, $(72, 128)$.