УДК 004.056

Экспериментальное исследование эффективности тестов для проверки генераторов случайных чисел

А. И. Миненко

Проведено исследование эффективности тестов “Стопка книг” и 15 тестов NIST для проверки генераторов случайных чисел. Протестированы 18 линейных конгруэнтных генераторов, RC4 и функция rand() в компиляторе C++ gcc 4.3.2 Linux. Показано, что “Стопка книг” может эффективнее находить отклонения от случайности, чем другие тесты.

Ключевые слова: тестирование случайных чисел, генераторы, стопка книг, тесты NIST.

1. Введение

Генераторы случайных чисел находят широкое применение в криптографии, в вычислительных методах и при имитационном моделировании. Задача генерирования последовательностей случайных чисел представляет большой интерес для разработчиков криптосистем. Во многих протоколах и шифрах случайные слова и числа используются как секретные ключи. Более того, с развитием криптографии выяснилось, что многие фундаментальные проблемы этой науки тесно связаны с генерированием и тестированием случайных чисел.

Одно из требований, предъявляемое к современным генераторам: генерируемая псевдослучайная последовательность должна быть статистически неотличима от абсолютно случайной. Методы, используемые для проверки этого условия, рассматриваются в рамках математической статистики. Таким образом, необходимо проверить гипотезу H_0 о том, что источник порождает символы алфавита равновероятно и независимо, против альтернативной гипотезы H_1, говорящей, что последовательность создана стационарным источником и H_0 не выполняется.

Эта задача привлекает внимание многих исследователей в силу её теоретической и практической важности. Так, Национальный институт стандартов и технологий США (NIST) разработал пакет статистических тестов для проверки бинарных последовательностей, созданных либо аппаратными, либо программными генераторами случайных чисел. Кроме этих тестов, существуют и другие. Одним из них является статистический тест “Стопка книг”, который был предложен в [4, 13].

В данном исследовании были протестированы различные генераторы случайных чисел, среди которых можно выделить линейные конгруэнтные генераторы, потоковый криптографический генератор RC4 и функцию rand в ОС Linux. Эти генераторы часто используются в различных приложениях, и поэтому их качество представляет практический интерес для разработчиков. Результаты этой работы позволяют сделать вывод о применимости генераторов. Кроме того, на основе проведённого исследования был сделан вывод о том, какой из тестов находит отклонения на большем количестве генераторов и при меньшей длине последовательности. Этим тестом оказался “Стопка книг”.

В главах 2 и 3 будет дано описание тестов. В главе 4 приведено описание генераторов. Глава 5 содержит описание экспериментов. В главе 6 – результаты. Выводы и рекомендации по применению тестов и генераторов находятся в главе 7.

2. Описание теста "Стопка книг"

Пусть некоторый источник порождает буквы из алфавита $A = \{a_1, a_2, ..., a_S\}$, $S > 1$, и требуется по выборке $x_1, x_2, ..., x_n$ проверить гипотезу $H_0 : p\ a_1 = p\ a_2 = ... = p\ a_S = 1/S$ против альтернативной гипотезы H_1, являющейся отрицанием H_0.

При тестировании по предлагаемому методу буквы алфавита A упорядочены и занумерованы в соответствии с этим порядком от 1 до S. Причём этот порядок меняется после анализа каждого выборочного значения x_i следующим образом: буква x_i, которую мы обозначаем через *a*, получает номер 1, номера тех букв, которые были меньше, чем номер *a*, увеличиваются на 1, а у остальных букв номера не меняются. Для более формального описания этого преобразования обозначим через $v^t\ a$ номер буквы $a \in A$ после анализа $x_1, x_2, ..., x_{t-1}$, и пусть начальный порядок $v^1()$ на буквах A задан произвольно. Тогда нумерация после анализа x_t определяется следующим образом:

$$v^{t+1}(a) = \begin{cases} 1, & \text{если } x_t = a, \\ v^t\ a\ + 1, & \text{если } v^t\ a\ < v^t\ x_t\ , \\ v^t\ a\ , & \text{если } v^t\ a\ > v^t\ x_t \end{cases} \qquad (1)$$

как в стопке книг, если считать, что номер книги совпадает с положением в стопке. Книга извлекается и кладётся наверх. Её номер становится первым; книги, которые первоначально были над ней, сдвигаются вниз, а остальные остаются на месте. Основная идея метода – подсчитывается не частота встречаемости букв в выборке $x_1, x_2, ..., x_n$, а частота встречаемости номеров букв (при описанном упорядочивании). В том случае, когда выполнена гипотеза H_1, вероятность (и частота встречаемости в выборке) некоторых букв больше $1/S$, и их номера в среднем будут меньше, чем у букв с меньшими вероятностями. Другими словами, книги, к которым обращаются чаще, проводят в верхней части стопки значительно большее время, чем остальные. Следовательно, вероятность обнаружить требуемую книгу в верхней части стопки больше, чем в нижней. Если же выполнена гипотеза H_0, то, очевидно, вероятность появления в выборке буквы с любым номером равна $1/S$.

При применении описываемого теста множество всех номеров $1, ..., S$ заранее, до анализа выборки, разбивается на $r > 1$ непересекающихся частей $A_1 = 1, 2, ...k_1$, $A_2 = k_1 + 1, ..., k_2$, ..., $A_r = k_{r-1} + 1, ..., k_r$. Затем по выборке $x_1, x_2, ..., x_n$ подсчитывается количество номеров $v^t\ x_t$, принадлежащих подмножеству A_j, которое мы обозначим через n_j, $j = \overline{1, r}$. При выполнении гипотезы H_0 вероятность того, что $v^t\ x_i$ принадлежит множеству A_j, пропорциональна количеству его элементов, т.е. равна $|A_j| / S$, а появление каждого элемента x_i в выборке независимо. Затем по критерию χ^2 проверяется гипотеза H_0^*:

$$P\ v^t\ x_t\ \in A_j\ = \frac{|A_j|}{S} \qquad (2)$$

против альтернативной гипотезы $H_1^* = \neg H_0^*$. Очевидно, при выполнении исходной гипотезы H_0 выполняется H_0^*, и наоборот, при выполнении гипотезы H_1^* выполняется H_1. Поэтому применение описанного критерия корректно. При применении критерия χ^2 вычисляется величина

$$x^2 = \sum_{j=1}^{r} \frac{n_j - nP_j^0}{nP_j^0}^2, \tag{3}$$

где $P_j^0 = |A_j| / S$. Известно, что распределение случайной величины x^2 асимптотически приближается к распределению χ^2 с $r-1$ степенью свободы при выполнении H_0.

Необходимо отметить, что критерий применим, если объём выборки n удовлетворяет неравенству

$$np_j \geq 10, \ j = \overline{1,r}. \tag{4}$$

3. Описание тестов NIST

Пакет статистических тестов разработан Лабораторией информационных технологий (англ. Information Technology Laboratory), являющейся исследовательской организацией Национального института стандартов и технологий (NIST). В состав пакета входят 15 статистических тестов, целью которых является определение меры случайности бинарных последовательностей, порождённых либо аппаратными, либо программными генераторами случайных чисел. Эти тесты основаны на различных особенностях, присущих только неслучайным последовательностям (см. [11,12]). Во всех тестах, если вычисленное в ходе теста значение P-value < 0.01, то данная двоичная последовательность не является истинно случайной. В противном случае последовательность носит случайный характер. Ниже мы приводим краткое описание этих тестов.

3.1. Frequency (Monobit) Test

Этот тест определяет соотношения между нулями и единицами во всей двоичной последовательности. Цель – выяснить, действительно ли число нулей и единиц в последовательности приблизительно одинаковы, как это можно было бы предположить в случае истинно случайной бинарной последовательности. Тест оценивает, насколько близка доля единиц к 0.5. Таким образом, число нулей и единиц должно быть примерно одинаковым. Все последующие тесты проводятся при условии, что пройден данный тест.

3.2. Frequency Test within a Block

Этот тест определяет количество единиц внутри блока длиной M бит. Цель – выяснить, действительно ли частота повторения единиц в блоке длиной M бит приблизительно равна $M/2$, как можно было бы предположить в случае абсолютно случайной последовательности. Если принять $M = 1$, данный тест переходит в тест Frequency (частотный побитовый тест).

3.3. Runs Test

Этот тест подсчитывает полное число серий в исходной последовательности, где под серией подразумевается непрерывная подпоследовательность одинаковых битов. Серия длины *k* бит состоит из *k* абсолютно идентичных битов, начинается и заканчивается с бита, содержащего противоположное значение. Цель данного теста – сделать вывод о том, действительно ли число серий, состоящих из единиц и нулей, с различными длинами соответствует их числу в случайной последовательности. В частности, определяется, быстро либо медленно чередуются единицы и нули в исходной последовательности.

3.4. Test for the Longest Run of Ones in a Block

В данном тесте определяется самая длинная серия единиц внутри блока длиной *m* бит. Цель – выяснить, действительно ли длина такой серии соответствует ожиданиям длины самой протяжённой серии единиц в случае абсолютно случайной последовательности. Следует заметить, что из предположения о примерно одинаковой частоте появления единиц и нулей (тест Frequency) следует, что точно такие же результаты данного теста будут получены при рассмотрении самой длинной серии нулей. Поэтому измерения можно проводить только с единицами.

3.5. Binary Matrix Rank Test

Здесь производится расчёт рангов непересекающихся подматриц, построенных из исходной двоичной последовательности. Целью этого теста является проверка на линейную зависимость подстрок фиксированной длины, составляющих первоначальную последовательность.

3.6. Discrete Fourier Transform (Spectral) Test

В данном тесте оценивается высота пиков дискретного преобразования Фурье исходной последовательности. Цель – выявление периодических свойств входной последовательности, например, близко расположенных друг к другу повторяющихся участков. Тем самым это явно демонстрирует отклонения от случайного характера исследуемой последовательности. Идея состоит в том, чтобы число пиков, превышающих пороговое значение в 95 % по амплитуде, было значительно больше 5 %.

3.7. Non-overlapping Template Matching Test

В данном тесте подсчитывается количество заранее определённых шаблонов, найденных в исходной последовательности. Цель – выявить генераторы случайных или псевдослучайных чисел, формирующие слишком часто заданные непериодические шаблоны. Как и в тесте № 8 на совпадение перекрывающихся шаблонов для поиска конкретных шаблонов длиной *m* бит используется окно также длиной *m* бит. Если шаблон не обнаружен, окно смещается на один бит. Если же шаблон найден, окно перемещается на бит, следующий за найденным шаблоном, и поиск продолжается дальше.

3.8. Overlapping Template Matching Test

Этот тест подсчитывает количество заранее определённых шаблонов, найденных в исходной последовательности. Как и в тесте № 7 на совпадение неперекрывающихся шаблонов для поиска конкретных шаблонов длиной *m* бит, используется окно также длиной *m* бит. Сам

поиск производится аналогичным образом. Если шаблон не обнаружен, окно смещается на один бит. Разница между этим тестом и тестом № 7 заключается лишь в том, что если шаблон найден, окно перемещается только на бит вперед, после чего поиск продолжается дальше.

3.9. Maurer's "Universal Statistical" Test

Здесь определяется число бит между одинаковыми шаблонами в исходной последовательности (мера, имеющая непосредственное отношение к длине сжатой последовательности). Цель теста – выяснить, может ли данная последовательность быть значительно сжата без потерь информации. В случае если это возможно сделать, то она не является истинно случайной.

3.10. Linear Complexity Test

В основе теста лежит принцип работы линейного регистра сдвига с обратной связью (англ. Linear Feedback Shift Register, LFSR). Цель – выяснить, является ли входная последовательность достаточно сложной для того, чтобы считаться абсолютно случайной. Абсолютно случайные последовательности характеризуются длинными линейными регистрами сдвига с обратной связью. Если же такой регистр слишком короткий, то предполагается, что последовательность не является в полной мере случайной.

3.11. Serial Test

Данный тест подсчитывает частоты всех возможных перекрываний шаблонов длины m бит на протяжении исходной последовательности битов. Целью является определение, действительно ли количество появлений 2^m перекрывающихся шаблонов длиной m бит приблизительно такое же, как в случае абсолютно случайной входной последовательности бит. Последняя, как известно, обладает однообразностью, то есть каждый шаблон длиной m бит появляется в последовательности с одинаковой вероятностью. Стоит отметить, что при m=1 тест на периодичность переходит в частотный побитовый тест (Frequency).

3.12. Approximate Entropy Test

Как и в тесте на периодичность, в данном тесте акцент делается на подсчёте частоты всех возможных перекрываний шаблонов длины m бит на протяжении исходной последовательности битов. Цель теста – сравнить частоты перекрывания двух последовательных блоков исходной последовательности с длинами m и m+1 с частотами перекрывания аналогичных блоков в абсолютно случайной последовательности.

3.13. Cumulative Sums (Cusum) Test

Тест заключается в максимальном отклонении (от нуля) при произвольном обходе, определяемым кумулятивной суммой заданных (-1, +1) цифр в последовательности. Цель данного теста — определить, является ли кумулятивная сумма частичных последовательностей, возникающих во входной последовательности, слишком большой или слишком маленькой по сравнению с ожидаемым поведением такой суммы для абсолютно случайной входной последовательности. Таким образом, кумулятивная сумма может рассматриваться как произвольный обход. Для случайной последовательности отклонение от произвольного обхода должно быть вблизи нуля. Для некоторых типов последовательностей, не являющихся в

полной мере случайными, подобные отклонения от нуля при произвольном обходе будут достаточно существенными.

3.14. Random Excursions Test

Этот тест подсчитывает число циклов, имеющих строго k посещений при произвольном обходе кумулятивной суммы. Произвольный обход кумулятивной суммы начинается с частичных сумм после последовательности (0,1), переведённой в соответствующую последовательность (-1, +1). Цикл произвольного обхода состоит из серии шагов единичной длины, совершаемых в случайном порядке. Кроме того, такой обход начинается и заканчивается на одном и том же элементе. Цель данного теста – определить, отличается ли число посещений определённого состояния внутри цикла от аналогичного числа в случае абсолютно случайной входной последовательности. Фактически данный тест есть набор, состоящий из восьми тестов, проводимых для каждого из восьми состояний цикла: -4, -3, -2, -1 и +1, +2, +3, +4.

3.15. Random Excursions Variant Test

В этом тесте подсчитывается общее число посещений определённого состояния при произвольном обходе кумулятивной суммы. Целью является определение отклонений от ожидаемого числа посещений различных состояний при произвольном обходе. В действительности этот тест состоит из 18 тестов, проводимых для каждого состояния: -9, -8, …, -1 и +1, +2, …, +9. На каждом этапе делается вывод о случайности входной последовательности.

4. Описание генераторов

Одними из наиболее распространённых являются линейные конгруэнтные генераторы. Они часто используются в библиотеках компиляторов и в других различных приложениях, где нужно просто и быстро получить случайные числа. Эти генераторы вычисляются по формуле:

$$X_{n+1} = aX_n + c \mod m,$$

где a, c, m, X_0 – параметры метода. Для всех, тестируемых здесь, линейных генераторов $X_0 = 1$. Полное описание дано в [1]. Будем обозначать эти генераторы через LCG(m, a, c). Информация о линейных генераторах взята из [2, 6, 7, 8]. Приведём список линейных генераторов, которые протестированы:

1) $LCG(2^{24}, 16598013, 12820163)$, этот генератор используется в Microsoft VisualBasic 6.0.
2) $LCG(2^{31}, 65539, 0)$, RANDU долгое время использовался во многих компьютерах в 1960-х и 1970-х годах.
3) $LCG(2^{32}, 1099087573, 0)$, этот генератор предложил Fishman.
4) $LCG(2^{32}, 69069, 1)$, этот генератор предложил Marsaglia.
5) $LCG(2^{32}, 69069, 5)$, использовался в компиляторах GNU.
6) $LCG(2^{32}, 1664525, 1013904223)$, предложен в Numerical Recipes.
7) $LCG(2^{32}, 22695477, 1)$, используется в Borland C/C++.
8) $LCG(2^{32}, 1103515245, 12345)$, используется в Digital Mars.
9) $LCG(2^{32}, 134775813, 1)$, используется в Borland Delphi.
10) $LCG(2^{32}, 214013, 2531011)$, используется в Microsoft Visual/Quick C/C++.

11) $LCG(2^{46},5^{13},0)$, использовался для аэродинамического моделирования в НАСА в исследовательском центре Ames.

12) $LCG(2^{48},25214903917,11)$, это генератор drand48 из стандартной библиотеки Unix.

13) $LCG(2^{48},5^{19},0)$, это традиционный генератор, использующийся в Национальной лаборатории США Los Alamos.

14) $LCG(2^{48},33952834046453,0)$, это один из генераторов, который предложил Fishman.

15) $LCG(2^{48},44485709377909,0)$, использовался в системе CRAY.

16) $LCG(2^{59},13^{13},0)$, базовый генератор в математической библиотеке NAG, также он присутствует в векторной статистической библиотеке(VSL), которая находится в библиотеке математического ядра Intel.

17), 18) $LCG(2^{63},5^{19},1)$, $LCG(2^{63},9219741426499971445,1)$, рекомендовано использовать на будущее в Национальной лаборатории США Los Alamos.

В криптографических приложениях часто используется потоковый генератор RC4, автором которого является Рональд Ривест (Ronald Rivest). Алгоритм RC4 строится на основе параметризованного ключом генератора случайных битов с равномерным распределением. Основные преимущества шифра – высокая скорость работы и переменный размер ключа. Полное описание этого генератора можно найти в [5].

В языке программирования C++ одним из генераторов случайных чисел является функция rand, которая входит в стандартные библиотеки. В данном исследовании проверялась функция в компиляторе C++ gcc 4.3.2 операционной системы Linux. Этот генератор выдаёт числа от 0 до $2^{31}-1$ включительно. Полное описание можно найти в [14].

5. Описание экспериментов

Все тесты применялись для проверки генераторов случайных чисел, предложенных в [2, 5, 6, 8, 14]. Всего было протестировано 18 линейных конгруэнтных генераторов, RC4 и функция rand() в компиляторе C++ gcc 4.3.2 Linux. Все генераторы предназначены для создания равномерно распределённых целых чисел. Известно, что младшие знаки чисел, порождаемые линейными конгруэнтными генераторами, часто далеки от случайных [1]. Следуя этой рекомендации, из созданных генератором значений выделялись старшие 8 бит. Для генератора rand() также выделялись старшие 8 бит. RC4 выдавал по 8 бит.

Каждый генератор выдавал 100 последовательностей одинаковой длины. Если генератор создаёт действительно равномерно распределённые последовательности, то в среднем 1 последовательность из 100 может быть забракована при уровне значимости $\alpha = 0.01$.

Доверительный интервал можно вычислить с помощью критерия χ^2. Величина статистики x^2 должна быть меньше квантиля $\chi^2_{1-\alpha,r-1}$, где α – ошибка первого рода, $(r-1)$ – степень свободы распределения χ^2. В данном случае $r = 2$. Примем $\alpha = 0.01$. Тогда

$$x^2 = \frac{(n_1 - n\alpha)^2}{n\alpha} + \frac{(n_2 - n(1-\alpha))^2}{n(1-\alpha)}, \tag{5}$$

где n_1 – количество забракованных последовательностей;

n_2 – количество случайных последовательностей, равное $100 - n_1$;

n – общее количество протестированных последовательностей, равное 100;

α – ошибка первого рода.

Подставим числовые значения в выражение (5). Вместо n_2 сделаем замену $100 - n_1$ и приравняем всё выражение к квантилю $\chi^2_{0.99,1} = 6.63$.

$$x^2 = \frac{(n_1 - 1)^2}{1} + \frac{(100 - n_1 - 99)^2}{99} = 6.63 \quad . \tag{6}$$

Преобразовав и упростив выражение (6), получим:

$$1.0101 n_1^2 - 2.0202 n_1 - 5.6198 = 0 . \tag{7}$$

Найдём корни уравнения (7). Они равны $n_{1_1} = 3.5619$, $n_{1_2} = -1.5619$. Отрицательного количества забракованных последовательностей не может быть, поэтому левая граница интервала равна 0. Дробного количества забракованных последовательностей тоже не может быть, поэтому правую границу интервала примем равной 4. Таким образом, доверительный интервал для забракованных последовательностей равен от 0 до 4 включительно.

Если количество забракованных последовательностей не попадает в этот интервал, то это говорит о том, что генератор выдаёт последовательности, при данной длине выборки, статистически отличимые от случайных. В этом случае будем говорить, что генератор не прошёл испытания, то есть забракован.

Тестирования проводились для последовательностей, выдаваемых генераторами, от 2^8 до 2^{23} бит. Для некоторых генераторов тестирование было проведено при больших длинах последовательности и только некоторыми тестами.

Опишем параметры для тестов.

"Стопка книг". Последовательность $x_n \in 0,1$ разбивалась на блоки длины l и при тестировании рассматривалась как выборка из алфавита размера $S = 2^l$. Множество всех номеров букв алфавита A разбивалось на два подмножества: $A_1 = a_1, a_2, ...a_k$, $A_2 = a_{k+1}, ..., a_S$. Второе подмножество не хранилось в памяти компьютера.

Binary Matrix Rank Test. $M = Q = 32$.

Non-overlapping Template Matching Test. m – количество бит в каждом шаблоне. $m = 9$.

B - шаблон. $B = 000000001$.

Overlapping Template Matching Test. m – количество бит в каждом шаблоне. $m = 9$. В-шаблон. $B = 111111111$. $M = 1032$.

Linear Complexity Test. $M = 500$.

Serial Test. $m = \lfloor \log_2 n \rfloor - 3$, где n – количество бит во входной последовательности.

Approximate Entropy Test. $m = \lfloor \log_2 n \rfloor - 7$ до длины 2^{16} бит входной последовательности, где n – количество бит во входной последовательности. Начиная с 2^{16} бит, $m = 10$.

Для остальных тестов параметры выбирались по умолчанию.

6. Результаты

В табл. 1 приведены результаты тестирования, перечисленных выше генераторов случайных чисел. В этой таблице представлены длины последовательностей в битах, выдаваемые генератором, с которых начинаются первые отклонения от случайности, определяемые данным тестом. Если отклонения обнаружены, то с увеличением длины входной последовательности отклонения возрастают.

Тестом "Стопка книг" найдены отклонения от случайности на 11 генераторах. Тестом DFT – на 4 генераторах. Тестом Serial – на 3 генераторах.

Из результатов видно, что тест “Стопка книг” забраковал большее количество генераторов. Во многих случаях это сделано при меньшей длине последовательности. А в некоторых случаях отклонения находит только “Стопка книг”.

Таблица 1. Тестирование генераторов

Генератор/Тест	Стопка книг	1.Frequency	2. Frequency Block	3. Runs	4.Longest Run	5. Binary Matrix	6. DFT	7. Non-overlapping	8. Overlapping	9. Mauerer”s “Universal”	10.Linear Complexity	11. Serial	12. Approximate Entropy	13. Cumulative Sums	14. Random Excursions	15. Random Exc Variant
$LCG(2^{24},16598013,12820163)$	2^{16}						2^{21}					2^{23}				
$LCG(2^{31},65539,0)$	2^{13}						2^{22}					2^{20}				
$LCG(2^{32},1099087573,0)$	2^{20}						2^{23}					2^{23}				
$LCG(2^{32},69069,1)$	2^{20}															
$LCG(2^{32},69069,5)$	2^{20}															
$LCG(2^{32},1664525,1013904223)$	2^{23}						2^{23}									
$LCG(2^{32},22695477,1)$	2^{20}															
$LCG(2^{32},1103515245,12345)$	2^{23}															
$LCG(2^{32},134775813,1)$	2^{20}															
$LCG(2^{32},214013,2531011)$	2^{19}															
$LCG(2^{46},5^{13},0)$																
$LCG(2^{48},25214903917,11)$																
$LCG(2^{48},5^{19},0)$																
$LCG(2^{48},33952834046453,0)$																
$LCG(2^{48},44485709377909,0)$																
$LCG(2^{59},13^{13},0)$																
$LCG(2^{63},5^{19},1)$																
$LCG(2^{63},921974142649997145$																
RC4	2^{32}															
rand (C++ gcc 4.3.2)																

7. Обсуждение и рекомендации

Проведённые исследования позволяют дать следующие выводы и рекомендации по применению рассмотренных тестов и генераторов.

Тест “Стопка книг” может эффективнее находить отклонения от случайности, чем тесты NIST. Рекомендуемые параметры для “Стопки книг”: длину блока l рекомендуется выбирать из ряда 8, 16, 20, 24, 32, 40 и так далее; размер одного или нескольких подмножеств $|A_i| = b \cdot \sqrt{2^l}$, где b – число из ряда 2, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 128, 160 и так далее.

В [11] для теста Approximate Entropy параметр m определялся в соответствии с неравенством $m < \log_2(n) - 2$, где n – количество бит во входной последовательности. При $m = \log_2(n) - 3$ все генераторы, перечисленные выше, бракуются при длине входной последовательности 2^{11} бит. Кроме того, на 2^{11} бит бракуются последовательности, созданные утилитой /dev/random в Linux, которую рекомендуется использовать для криптографических приложений. Эта утилита создаёт случайные числа, основным источником которых являются драйвера устройств компьютера [9]. В [12] неравенство для параметра m было изменено на $m < \log_2(n) - 5$. При $m = \log_2(n) - 6$ все генераторы бракуются при длине входной последовательности 2^{21} бит, а $LCG(2^{24}, 16598013, 12820163)$ на 2^{25}, что не согласуется с результатами, полученными при помощи других тестов. Кроме того, на 2^{21} бит этим тестом бракуются последовательности, записанные в файлах, предложенные Marsaglia в “The Marsaglia Random Number CDROM” (bits.01, …). Эти файлы сформированы с помощью лучших псевдослучайных генераторов вместе с тремя источниками белого шума [10]. На основе выше изложенного сделан вывод о том, что тест Approximate Entropy с такими параметрами даёт недостоверный результат.

Проведённое исследование позволяет дать рекомендации по применению генераторов случайных чисел, которые были протестированы.

Первые 10 генераторов, представленных в табл. 1, не рекомендуется использовать в современных приложениях. RC4 рекомендуется использовать для создания последовательностей длиной до 2^{32} бит. Остальные генераторы можно использовать, так как до 2^{23} бит не было найдено отклонений выше перечисленными тестами. Однако линейные генераторы не рекомендуется использовать для криптографических приложений [3].

Литература

1. Кнут Д. Искусство программирования на ЭВМ. Т. 2. Получисленные алгоритмы. М.: Мир, 1977.
2. Линейный конгруэнтный метод. // Википедия: свободная энциклопедия. [Электронный ресурс]. URL: `http://ru.wikipedia.org/wiki/Линейный_конгруэнтный_метод` (дата обращения: 15.11.2010).
3. Монарёв В. А. Построение новых статистических тестов и их применение в криптографии. Диссертация. Новосибирск, 2005.
4. Рябко Б. Я., Пестунов А. И. “Стопка книг” как новый статистический тест для случайных чисел. Проблемы передачи информации. Том 40. Вып. 1, 2004.
5. Рябко Б. Я., Фионов А. Н. Криптографические методы защиты информации: Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2005. – 229 с.: ил.
6. Karl Entacher A collection of classical pseudorandom number generators with linear structures – advanced version. June 16, 2000. [Электронный ресурс]. URL: `http://random.mat.sbg.ac.at/results/karl/server/server.html` (дата обращения: 15.11.2010).
7. L'Ecuyer P. Tables of linear congruental generators of different sizes and good lattice structure. Math. Comput. 1999. V.68. P. 249-260.
8. L'Ecuyer P. and Simard R., TestU01: A C Library for empirical testing of random number generators, ACM Transactions on Mathematical Software, 2007.
9. Linux Programmer's Manual. [Электронный ресурс]. URL: `ttp://www.kernel.org/doc/man-pages/online/pages/man4/random.4.html` (дата обращения: 15.11.2010).
10. Marsaglia G. The Marsaglia Random Number CDROM, 1995. [Электронный ресурс]. URL: `http://www.stat.fsu.edu/pub/diehard/cdrom/` (дата обращения: 15.11.2010).

11. Rukhin A. and others A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 Revision 1 August 2008.
12. Rukhin A. and others A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 Revision 1a April 2010.
13. Ryabko B. Ya., Monarev V. A. Using information theory approach to randomness testing. Journal of Statistical Planning and Inference, 2005, v. 133, n.1, pp. 95-110.
14. The GNU C Library. [Электронный ресурс]. URL: `http://www.gnu.org/software/libc/manual/` (дата обращения: 15.11.2010).

Статья поступила в редакцию 04.10.2010;
переработанный вариант — 22.11.2010

Миненко Андрей Иванович
магистрант СибГУТИ (630102, Новосибирск ул. Кирова 86), e-mail: minenkoai@mail.ru.

Experimental research of efficiency of tests for random number generators

A. I. Minenko

The efficiency of the “Book stack” test and 15 tests of the NIST for random number generators are investigated. 18 linear congruent generators, RC4 and function rand() in the compiler C++ gcc 4.3.2 Linux have been tested. It is shown that the “Book stack” can find a deviation from randomness more effectively than other tests.

Keywords: random number testing, random number generator, book stack, NIST tests.