TestU01: библиотека C для эмпирического тестирования генератор случайных чисел

Р. А. Сафаров

Новосибирский государственный университет

2021

Научный руководитель – д-р. техн. наук, проф Б. Я. Рябко

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.
Перед ответственным использованием в математическом моделировании и в криптографии программные генераторы ПСП должны быть протестированы.

Введение

Для тестирования последовательностей на случайность существует большое количество алгоритмов, а для удобства проверки последовательностей уже реализованы программные продукты, содержащие в себе некоторые наборы тестов. Среди них наиболее распространены тесты **DieHard**, **NIST STS**, **TestU01, PractRand, gjrand, Стопка книг** и другие.

Введение

В данном исследовании проведено применять практически пакет TestU01 к тестированию генератор в C++ и исследование эффективности тестов **Стопка книг**, **TestU01** и тестов **PractRand** для проверки генераторов случайных чисел. На основе проведенного исследования был сделан вывод о том, какой из тестов находит отклонения на большем количестве генераторов и при меньшей длине последовательности.

Протестированы 14 различных генераторы случайных чисел, среди которых можно выделить линейные конгруэнтные генераторы, потоковый криптографический генератор RC4, функцию rand в ОС Linux, Mersenne Twister и AES+Feistel+Reverie.

Тесты

Существуют различные тесты, которые оценивают, насколько исследуемая последовательность бит «похожа» или «не похожа» на действительно случайную. Методы оценки качества генераторов случайных и псевдослучайных последовательностей можно разделить на две группы:

1. **Графические тесты**
2. **Статистические тесты**

Графические тесты

Свойства последовательностей отображаются в виде графических зависимостей, по виду которых делают выводы о свойствах исследуемой последовательности.

К данной категории можно отнести следующие тесты:

1. **Гистограмма распределения элементов последовательности;**
2. **Распределение на плоскости**
3. **Проверка серий**
4. **Проверка на монотонность**
5. **Автокорреляционная функция**
6. **Профиль линейной сложности**
7. **Графический спектральный тест**

Статистические тесты

В отличие от графических тестов, статистические тесты выдают численную характеристику последовательности и позволяют однозначно сказать, пройден тест или нет.

Наиболее известны следующие программные пакеты статистических тестов:

1. **Тесты NIST**
2. **TEST-U01**
3. **CRYPT-X**
4. **DIEHARD**
5. **ENT**
6. **PractRand**
7. **Стопка книг**

Описание теста “TestU01”

TestU01 — это пакет статистических эмпирических тестов, реализованный на языке ANSI C, который предлагает набор утилит для тестирования генераторов случайных чисел. Он предлагает четыре группы модулей для работы с генераторами:

1. реализация заранее запрограммированного генератора (**модуль u**)
2. реализация специальных статистических тестов (**модуль s**);
3. реализация батарей статистических тестов (**модуль b**);
4. применение тестов на целых семействах генераторов (**модуль f**).

Описание теста “TestU01”

В пакете **TestU01** содержится шесть батарей статистических тестов.

1. **SmallCrush** (состоит из 10 тестов)
2. **Crush** (состоит из 96 тестов)
3. **BigCrush** (состоит из 106 тестов)
4. **Alphabit** (состоит из 17 тестов)
5. **Rabbit** (состоит из 38 тестов)
6. **BlockAlphabit**(состоит из 17 тестов)

Описание теста “PractRand”

PractRand - самый простой в использовании и удобный для измерения «эффективности». Он принимает на вход поток байт, может тестировать 32 и 64 битные генераторы. Способен справляться с очень большими объемами данных.

Описание теста “Стопка книг”

Основная идея метода а упарядочивании элементов как в стопке книг. Книга вынимается из стопки и кладётся наверх. Её номер становится первым; книги, которые до этого были над ней двигаются вниз, а остальные остаются на месте. Пусть некоторый источник порождает буквы из алфавита $A = (a_1, a_2, \ldots, a_s)$, $S > 1$ и требуется по выборке $x_1, x_2, \ldots, x_n$ проверить гипотезу:

$$H_0 : p(a_1) = p(a_2) = \ldots = p(a_s) = 1/S$$

Против альтернативной гипотезы H_1, являющейся отрицанием H_0

Описание теста “Стопка книг”

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 как в стопке книг.

Обозначим через $b^t(a)$ номер буквы $a \in A$

$\{1,..., S\}$

При применении описываемого теста множество всех номеров $A_1 = \{1, 2, ..., k_1\}, A_2 = \{k_1 + 1, ..., k_2\}, ...,$ $..., A_r = \{k_{r-1} + 1, ..., k_r\}$, заранее до анализа выборки, разбивается на r>1 непересекающихся частей

$x_1, x_2, ..., x_n$

$b^t(x_t)$ A_j

Затем по выборке $n_j, j = \overline{1, r}.$ подсчитывается количество номеров , принадлежащих подмножеству которое мы обозначим через

Описание теста “Стопка книг”

При выполнении гипотезы H_0 вероятность того, что номер принадлежит множеству A_j пропорциональна количеству элементов а этом подмножестве, т.е. равно A_j / S. Затем по критерию x^2 проверяется гипотеза

$$H_0^* : P\{b^t(x_t) \in A_j\} = A_j / S.$$

против альтернативной гипотезы $H_1^* = \neg H_0^*$. Очевидно, при выполнении исходной гипотезы H_0 выполняется H_0^* и наоборот, при выполнении H_1^* гипотезы выполняется H_1.

Описание генераторов

1. *LCG*(224 ,16598013,12820163), этот генератор используется в Microsoft VisualBasic 6.0.
2. *LCG*(231,65539,0), RANDU долгое время использовался во многих компьютерах в 1960-х – 1970-х годах,
3. *LCG*(232 ,1099087573,0), этот генератор предложил Fishman.
4. *LCG*(232 ,69069,1), этот генератор предложил Marsaglia.
5. *LCG*(232 ,69069,5), использовался в компиляторах GNU.
6. *LCG*(232 ,1664525,1013904223), предложен в Numerical Recipes.
7. *LCG*(232 , 214013, 2531011), используется в Microsoft Visual/Quick C/C++.
8. *LCG*(246 ,513,0), использовался для аэродинамического моделирования в НАСА в исследовательском центре Ames.
9. *LCG*(263,519 ,1), LGG (263, 92197414264999714445, 1) рекомендовано использовать на будущее а Национальной лаборатории США Los Alamos.

Описание генераторов

10. LCG(263,92197414264999714 4
11. RC4
12. rand (C++ gcc 4.3.2)
13. Mersenne twister
14. AES+Feistel+Reverie

Описание экспериментов

Каждый генератор выдавал 100 последовательностей одинаковой длины. В среднем 1 последовательность из 100 может быть забракована при уровне значимости a= 0.01. Доверительный интервал, вычисленный с помощью χ^2 критерия равен [0;4]. Если количество забракованных последовательностей не попадает а этот интервал, то это говорит о том, что генератор выдаёт последовательности, при данной длине выборки, статистически отличимые от случайных. В этом случае будет 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

Описание экспериментов

Тестом "**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 позиций а "Стопке книг" развивалось на два подмножества $A_1 = \{a_1, a_2, ..., a_k\}, A_2 = \{a_{k+1}, ..., a_s\}$
Второе подмножества не хранилось в памяти компьютера.
При исследовании тестами **TestU01, PracRand**, выбирались рекомендуемые параметры.

Результаты

В данной работе приведены результаты тестирования, перечисленных выше генераторов случайных чисел и 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.

В результате было показано, что тестом

1. **PractRand** (стандартный, 1 терабайт) на 10 генераторах
2. **TestU01**(SmallCrush, Crush, BigCrush, Alphabit, Rabbit) на 9 генераторах
3. **Стопка книг** на 9 генераторах найдены отклонения от случайности

Результаты

Генератор/Тест	Стопка книг	SmallCrush	Crush	BigCrush	Alphabit	Rabbit	PractRand
$LCG(2^{24}, 16598013, 12820163)$	2^{16}		2^{23}	2^{13}			2^{12}
$LCG(2^{31}, 65539, 0)$	2^{13}			2^{21}			2^{20}
$LCG(2^{32}, 1099087573, 0)$	2^{20}		2^{24}	2^{22}			2^{21}
$LCG(2^{32}, 69069, 1)$	2^{20}			2^{23}			2^{22}
$LCG(2^{32}, 69069, 5)$	2^{20}			2^{23}			2^{22}
$LCG(2^{32}, 1664525, 1013904223)$	2^{23}			2^{23}			2^{22}
$LCG(2^{32}, 214013, 2531011)$	2^{19}			2^{24}			2^{23}
$LCG(2^{46}, 5^{13}, 0)$				2^{25}			2^{24}
$LCG(2^{63}, 5^{19}, 1)$							2^{32}
$LCG(2^{63}, 9219741426499971 44,$							2^{34}
RC4	2^{32}						
rand (C++ gcc 4.3.2)							
Mersenne twister	2^{32}						
AES+Feistel+Reverie							

Заключение

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

1) Тест **Prandrand, BigCrush, Стопка книг** может эффективнее находить отклонения от случайности, чем другие тесты, так как он нашёл отклонения на большем количестве генераторов. Во многих случаях это сделано при меньшей длине последовательности.

2) Линейные генераторы с параметром $m \leq 2^{32}$не рекомендуется использовать в современных приложениях. RC4 рекомендуются использовать для создания последовательности длиной до 2^{23} бит. Остальные генераторы можно использовать, так как до 2^{32}. Не было найдено отключений выше перечисленными тестами

Заключение

3) Рекомендуемые параметры для "**Стопки книг**" l длину блока рекомендуется выбирать из ряда 8, 16, 20, 32, 40 и так далее; размер одного или нескольких подмножеств $4 \geq b * \sqrt{2^l}$ где b число из ряда 2, 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 128, 160 и так далее.

4) С помощью **PractRand** и **TestU01**, как правило, легче всего интерпретировать результат.

5) **PractRand** требовал больше битов ввода для тестирования, чем другие наборы тестов - это может быть проблемой, если ваш RNG очень медленный или иным образом ограничен по объему производимых данных.

Спасибо за внимание!