Avaliação de Geradores de Números Pseudo-Aleatórios

Vera Lúcia da Silva[1], Juliano de Almeida Monte-Mor[1], Nei Yoshihiro Soma[1]

[1] Divisão de Ciência da Computação – Instituto Tecnológico de Aeronáutica (ITA)
12228-901 – São José dos Campos – SP – Brazil

{verals, montemor, nysoma}@ita.br

Resumo. *A geração de números aleatórios é de fundamental importância para o processo da segurança de dados, desta forma, a escolha de um gerador de números pseudo-aleatórios não pode ser realizada ao acaso. Faz-se necessário o uso de conjuntos de testes estatísticos para auxiliar nessa atividade. Este trabalho descreve os testes estatísticos do Sistema DIEHARD, um rigoroso conjunto de testes para geradores de números pseudo-aleatórios. Foram implementados e analisados alguns geradores das linguagens de programação mais utilizadas e outros tradicionalmente conhecidos, visando avaliar seu grau de aleatoriedade.*

1. Introdução

Números aleatórios, dentre outras utilidades, são usados para inserir dados imprevisíveis e não-determinísticos em algoritmos para comunicação segura entre computadores, principalmente em protocolos de autenticação, na geração de assinaturas digitais e de chaves usadas na criptografia.

Entretanto, os geradores de números pseudo-aleatórios produzem números por meio da execução repetitiva de um algoritmo. Como não são capazes de gerar números verdadeiramente aleatórios, sua saída pode ser previsível.

É importante que o grau de aleatoriedade e a independência dos valores gerados sejam garantidos para que os resultados sejam satisfatórios, assegurando a autenticação, a confidencialidade e a integridade das informações compartilhadas.

Desta forma, como já ressaltado por Donald Knuth, "um gerador de números pseudo-aleatórios não pode ser escolhido aleatoriamente" [Knuth 1998], faz-se necessário identificar suas características, avaliando sua performance, aleatoriedade e o grau de independência dos valores gerados.

A existência de conjuntos de testes estatísticos objetiva auxiliar nessa atividade. A teoria dos testes estatísticos fornece algumas medidas quantitativas para aleatoriedade. Entretanto, não há um número exato de testes que devem ser realizados para provar que uma seqüência é verdadeiramente aleatória. Quanto mais testes forem executados, maior será a certeza de aleatoriedade da seqüência.

Dentre os conjuntos de testes estatísticos mais difundidos é possível citar: Diehard, Crypt-XS, NIST Statistical Test Suíte. Neste trabalho foram pesquisados os testes estatísticos do Sistema Diehard, por ser esses considerados mais rigorosos, cf. e.g. [Knuth 1998]. Eles foram utilizados para avaliar alguns dos geradores de números pseudo-aleatórios mais utilizados e conhecidos, na tentativa de verificar se são

adequados para a geração de números imprevisíveis e não-determinísticos necessários para aplicações seguras.

Os assuntos abordados neste artigo estão divididos em quatro seções. Na seção 2 são descritos os testes do sistema Diehard. A seção 3 demonstra a análise e os resultados obtidos com a aplicação do conjunto de testes Diehard aos principais geradores de números pseudo-aleatórios. E finalmente, a seção 4 tece as considerações finais deste trabalho.

2. O Sistema Diehard

O Sistema Diehard foi desenvolvido por George Marsaglia, professor e pesquisador do Departamento de Estatísticas e Pesquisas em Computação da Universidade Flórida. Ele consiste num conjunto de 15 testes estatísticos que procuram identificar padrões e distribuições não-uniformes em seqüências aleatórias, e.g. [Marsaglia 1984, 2004].

A idéia principal de cada teste é sumarizada a seguir e na seqüência é apresentado o resumo dos testes estatísticos utilizados (Tabela 1):

(1) Espaçamento entre Aniversários: o nome vem do bem conhecido problema de se determinar quantas pessoas são necessárias para que pelo menos duas delas tenham a mesma data de aniversário, com probabilidade não inferior a 50%, que para o caso é de vinte e três pessoas. A idéia do teste é o de se considerar como data de nascimentos, números de 8 bits e testar o espaçamento entre duas ocorrências.

(2) Permutação com 5 Elementos e Sobreposição: Utiliza-se uma seqüência de 10^6 inteiros de 32 bits. Cada conjunto de 5 inteiros consecutivos desta seqüência é classificado de acordo com as 5! inversões de paridade entre as permutações. Procede-se a uma inserção de um dígito mais à direita e eliminação do mais à esquerda (sobreposição). Devido a essa sobreposição, mostra-se que em termos estatísticos as classes de inversões passam de 5! para 5!-21. O teste é executado duas vezes, tem-se então a geração de duas seqüências de 10^6 inteiros.

(3) Posto de Matrizes 31 x 31 e 32 x 32: Consideram-se 40.000 matrizes de 31 x 31 e 32 x 32 que são divididas de acordo com os postos das mesmas, cada um dos elementos das matrizes é formado a partir de números de 31 (32) bits, sendo portanto matrizes com entradas 0 ou 1. Pode-se mostrar que para tais matrizes e com entradas aleatórias, os postos das mesmas são, via de regra, maiores ou iguais que 29. O teste considera a freqüência de ocorrência dos postos de 29 à 31 (ou 32).

(4) Posto de Matrizes 6 x 8: A exemplo do caso anterior as matrizes (100.000 delas) são obtidas a partir de seis números inteiros aleatórios de 32 bits. De cada um destes números um dado byte específico é escolhido. Assim, uma linha da matriz é formada pelos 8 bits retirados destes 6 números. Procede-se então ao cálculo do posto da matriz resultante.

O *Teste dos Macacos* é utilizado nos testes *(5)* e *(6)*, sendo que a idéia é a de simular a digitação de um macaco que ao utilizar um dado teclado, o faz de maneira aleatória. A "produção literária" resultante da digitação do macaco é avaliada em termos de aleatoriedade, e.g. [Marsaglia 1993].

(5) Fluxo de Bits: o arquivo de entrada é visto como um fluxo de bits, da forma b1, b2, etc. Considera-se um alfabeto binário onde cada palavra é composta por 20

letras. Assim, a primeira palavra é b1b2...b20, a segunda é b2b3...b21, e assim por diante. O teste conta o número de palavras de 20 letras ausentes numa seqüência de 2^{21} palavras de 20 letras sobrepostas (isso equivale a 2^{21} + 19 bits, que corresponde ao tamanho da cadeia analisada).

(6) Sobreposição de Pares, Quádruplos e DNA: No teste *Sobreposição de Pares* são contadas as palavras com 2 letras de um alfabeto de 2^{10} letras, que não aparecem numa seqüência de 2^{21} palavras sobrepostas. O teste *Sobreposição de Quádruplos* é similar ao anterior, entretanto, nele são consideradas palavras de 4 letras de um alfabeto com 2^5 letras. No *DNA* o alfabeto tem 4 letras e as palavras possuem 10 letras. Pode-se mostrar que para todos os casos, a quantidade média de palavras "perdidas" tem comportamento igual para os três casos, isto é, distribuição normal com igual média e igual desvio padrão.

(7) Quantidade de 1's em Fluxo de Bytes: Dado um arquivo de inteiros de 32 bits examina-o como uma seqüência de bytes. Cada byte deve ocorrer por sua vez, com probabilidades iguais respectivamente a 1, 8, 28, 56, 70, 56, 28, 8, 1 de um total de 256 possibilidades. São contadas as ocorrências de cada palavra numa seqüência de 256.000 palavras contendo 5 letras sobrepostas.

(8) Quantidade de 1's para um Byte Específico: Como no teste anterior, considera-se um arquivo de entrada como uma seqüência de números inteiros de 32 bits. Para cada inteiro, um byte específico é escolhido, e.g., o mais à esquerda do inteiro com 32 bits. Cada byte pode conter de 0 a 8 números 1 (8 bits), com probabilidades iguais respectivamente a 1, 8, 28, 56, 70, 56, 28, 8, 1 sobre um total de 256 possibilidades. Esses bytes específicos produzem palavras de 5 letras, sendo que as palavras, a exemplo do teste anterior, também se sobrepõem, de modo que cada nova palavra é formada pelas últimas 4 letras da palavra anterior mais à próxima letra lida. As ocorrências são testadas numa seqüência de 256.000 palavras.

(9) Conjunto de Estacionamentos: Simula-se um estacionamento onde seja possível estacionar aleatoriamente um carro, representado por um círculo de raio 1. O objetivo do teste é estacionar sucessivamente os carros no estacionamento um a um. Cada tentativa de se estacionar um carro conduz a uma colisão (outro carro na posição escolhida) ou a um sucesso, onde cada sucesso acarreta um incremento à lista dos carros já estacionados. Pode-se mostrar que a distribuição da quantidade de sucessos é uma dada distribuição normal.

(10) Distância Mínima em Quadrado: Nesse teste escolhem-se 8000 pontos aleatórios em um quadrado de lado 10.000. São determinadas as distâncias mínimas entre todos os $(n^2 - n) / 2$ pares de pontos. Pode-se mostrar que se a distribuição dos pontos for realmente aleatória, então d^2, o quadrado da distância mínima obedece à uma dada distribuição exponencial.

(11) Distância em Esferas: São escolhidos 4000 pontos aleatórios num cubo de aresta 1000. Cada ponto é considerado como sendo o centro de uma esfera cujo raio é o segmento de reta desse centro com seu vizinho mais próximo. Pode-se mostra que o volume da menor destas esferas obedece a uma dada distribuição exponencial.

(12) Compressão: São utilizados valores de ponto flutuante no intervalo [0, 1), onde cada um desses é obtido a partir de um número inteiro do arquivo de testes. O método começa com um inteiro $k = 2^{31}$ e são determinadas quantas iterações (J) são

necessárias para reduzir k a 1 usando a "compressão" $k = \lceil kU \rceil$, onde U é um número do arquivo testado. Os valores J's são agrupados em 43 classes e as freqüências de ocorrência calculadas.

(13) Somas Sobrepostas: A exemplo do teste anterior são escolhidos inteiros com representação em ponto flutuante para que se obtenha uma seqüência U1, U2, ... de variáveis uniformes no intervalo [0, 1]. Calcula-se então as somas sobrepostas, S1 = U1 + … + U100; S2 = U2 + … + U101. É possível mostrar que os valores Si's podem ser transformados em uma distribuição uniforme.

(14) Corridas: Determina-se quantas seqüências (corridas) de uma dada sucessão ininterrupta de dígitos está em ordem crescente (ou decrescente). Uma corrida de comprimento k consiste em exatamente k dígitos em ordem crescente (ou decrescente). Por exemplo, as corridas |129|8530|789| têm duas corridas crescentes 129 e 789; e uma corrida decrescente 8530.

(15) Craps: Vem de um jogo em que cada lance corresponde ao arremesso de 3 dados. São jogadas 200.000 partidas e contabiliza-se tanto a quantidade de vitórias quanto o número de lançamentos de dados necessários para terminar cada jogo. Pode-se mostrar que a quantidade de vitórias é uma distribuição normal caso os dados não sejam viciados.

Tabela 1 – Diehard e seus respectivos Testes Estatísticos.

Teste Diehard	Teste Estatístico
1. Espaçamento entre Aniversários	$\chi 2$ com 1000 graus de liberdade
2. Permutação com 5 Elementos e Sobreposição	$\chi 2$ com 99 graus de liberdade
3. Posto de Matrizes 31x31 e 32x32	$\chi 2$ com 3 graus de liberdade
4. Posto de Matrizes 6x8	$\chi 2$ com 2 graus de liberdade
5. Fluxo de Bits	Distribuição Normal μ=141.909 e σ =428
6. Sobreposição de Pares, Quádruplos e DNA	Distribuição Normal μ=141.909 e σ = 290
7. Quantidade de 1's em Fluxo de Bytes	$\chi 2$ com 2500 graus de liberdade
8. Quantidade de 1's para um Byte Específico	$\chi 2$ com 2500 graus de liberdade
9. Conjunto de Estacionamentos	Distribuição Normal μ=3.523 e σ =21.9
10. Distância Mínima em Quadrado	Kolgomorov-Smirnov μ= 0,995
11. Distância em Esferas	Kolgomorov-Smirnov $\mu=40\pi$
12. Compressão	$\chi 2$ com 42 graus de liberdade
13. Somas Sobrepostas	Kolgomorov-Smirnov μ= 0,5
14. Corridas	Kolgomorov-Smirnov
15. Craps	$\chi 2$ com 21 graus de liberdade

Os testes do Diehard apresentam seus resultados baseados em análise estatística. Cada teste retorna um valor *p* (*p-value*), sendo este obtido por meio da função $p=F(X)$, onde F é a distribuição normal de amostras aleatórias da variável X.

O valor de p varia entre 0 e 1, onde valores próximos a estes limites indicam que a seqüência não apresenta um grau aceitável de aleatoriedade. Considera-se que se o valor de $p<0.025$ ou $p>0.975$, o gerador fracassou para o teste, e.g. [Marsaglia 2004]. No caso de um teste ter como resultado mais de um valor p, a falha foi considerada, neste trabalho, quando a quantidade de valores falhos foi maior que a quantidade de valores aceitáveis, e em caso de empate considerou-se que o gerador passou no teste.

3. Avaliação de Geradores

Neste trabalho foram testados e avaliados os seguintes geradores: *Rand* - gerador de 16 bits da biblioteca padrão da linguagem C/C++; *Random* - gerador da biblioteca padrão da linguagem Java; *Random* - gerador da biblioteca padrão da linguagem Delphi; *LRand* - gerador de 32 bits da linguagem C/C++, e.g. [Wichmann and Hill 1987]; *Ran3* - *Numerical Recipes,* e.g. [Press et al. 1992]; e *Blum-Blum-Shub* - muito utilizado na criptografia, e.g. [Rukhin et al. 2001].

Para cada gerador selecionado foram produzidos 10 arquivos de testes, os quais foram submetidos ao Sistema Diehard. Os resultados dos testes de cada arquivo foram contabilizados, identificando os testes para os quais os geradores falharam ou passaram. A Figura 1 apresenta o número de falhas nos 15 testes relacionados na Tabela 1 para os 10 arquivos de entrada.

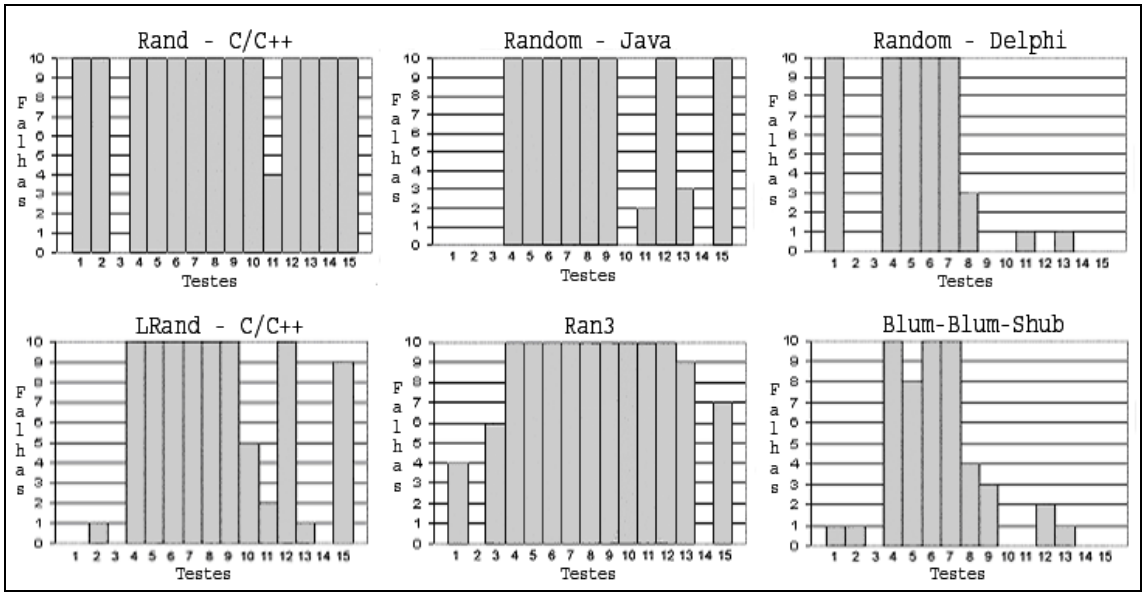

Figura 1. Avaliação dos Geradores.

Nos 10 arquivos de entrada todos os geradores falharam nos testes: Posto de Matrizes 6x8; Sobreposição de Pares, Quádruplos e DNA; e Quantidade de 1's em Fluxo de Bytes. Constatou-se que geralmente as falhas de um gerador sempre ocorrem nos mesmos testes.

Os resultados mostraram que para arquivos de testes diferentes, criados a partir de um mesmo gerador, os testes podem resultar em valores diferentes para p. Entretanto, a diferença dos valores de p é mínima e, em poucos casos, as seqüências de um mesmo gerador falham em testes diferentes.

O gerador *Blum-Blum-Shub* teve o melhor resultado nos testes, falhando em média em apenas 5 testes. O gerador *Rand* teve o pior resultado, falhando em média em 13.4 testes. Os geradores *Random* – Linguagem Delphi, *Random* – Linguagem Java, *LRand* e *Ran3,* falharam em média em 5.5, 8.5, 8.8 e 11.6 testes respectivamente.

4. Conclusões

Para a segurança de sistemas computacionais é importante assegurar a autenticação, a confidencialidade e a integridade dos dados compartilhados. A geração de números aleatórios é um fator chave para garantir estes elementos no processo de segurança.

Neste trabalho foram implementados e analisados 06 geradores de números pseudo-aleatórios. Foram criados para cada gerador 10 arquivos com seqüência de números aleatórios diferentes. Os resultados dos testes do Diehard foram analisados, contabilizando a quantidade de testes para os quais os geradores falharam.

Não é aconselhável a utilização dos geradores *Rand* – Linguagem C/C++ e *Ran3 - Numerical Recipes* em algoritmos para segurança de informações, pois os mesmos falharam em média em 89% e 77% dos testes estatísticos do Diehard respectivamente. Note-se que observação dessa natureza já havia sido apresentada em [Knuth 98] em relação a geradores congruenciais.

Dentre os geradores testados é possível destacar o gerador *Blum-Blum-Shub*, pois ele obteve sucesso em média em 66% dos testes estatísticos do Diehard, e também o gerador *Random* da biblioteca padrão da Linguagem Delphi, pois foi o melhor entre os geradores das linguagens, com média de 63% de sucesso nos testes.

Foi desenvolvido um ambiente gráfico integrado e amigável para o Sistema Diehard, visando tornar menos árdua a tarefa de avaliação de geradores de números pseudo-aleatórios. Este ambiente gráfico, bem como os relatórios técnicos produzidos, encontram-se disponíveis com os autores.

Referências

Knuth, D.E. “The Art of Computer Programming”, Third Edition. Volume 2: Seminumerical. Algorithms, Capítulo 3: Números Aleatórios (Pseudo), Addison-Wesley Publishing Company, Massachusetts, 1998.

Marsaglia, G. (2004) “The Marsaglia Random Number Cdrom including the Diehard Battery of Tests of Randomness”, http://stat.fsu.edu/~geo/, pesquisado em 10/06/2004.

Marsaglia, G. (1993) “Monkey Tests for Random Number Generators”, In: Computers & Mathematics with Applications, 9, 1--10.

Marsaglia, G. (1984) “A Current View of Random Number Generators”, Keynote Address, Computer Science and Statistics: 16th Symposium on the Interface, Atlanta. Published by Elsevier Press.

Press, William H., et al. “Numerical Recipes in C: The Art of Scientific Computing”, Cambridge University Press, 1988-1992.

Rukhin, A., et al. (2001) “A Statistical Test Suite for Random and Pseudorandom Number Generator for Cryptographic Applications”. In: Nist Special Publication 800-22.

Wichmann, B., Hill, D. (1987) “Three-Generator Random Number Algorithm”, In: Byte Magazine, March, pp. 127-8.