V Escola do CBPF – Rio de Janeiro, de 00-00 de 2004

Simulações Computacionais de Sistemas Complexos

T.J.P.Penna
Instituto de Física, Universidade Federal Fluminense,
Av. Litorânea, s/n — Boa Viagem
24210-340 Niterói, RJ,
email: tjpp@if.uff.br,
homepage: http://www.if.uff.br/~tjpp

RESUMO

Neste mini-curso vamos apresentar algumas das técnicas utilizadas em simulações de sistemas complexos. Não é nossa intenção revisar ou apresentar técnicas tradicionais de métodos numéricos. Boa parte dos temas escolhidos para este curso foi determinada pela audiência. Assim sendo, os estudantes que assistiram ao mesmo serão os responsáveis pelas falhas e imprecisões destas notas. Alguns exercícios e questões são propostos para o leitor. É recomendável, mas não imprescindível, que os mesmos sejam feitos antes de seguir na leitura do texto.

1 Sistemas Complexos

Muitos artigos científicos começam dizendo que o assunto ali tratado é de grande interesse porém complicado. Aqui, vamos começar dizendo que sistemas complexos desperta enorme interesse mas "complexo" não quer dizer complicado. A definição do que é um sistema complexo é, digamos, um tanto complexa. É como a definição do que é um ser vivo. Se usarmos as propriedades usuais de crescimento, metabolismo (energia-massa), movimento, reprodução e resposta a estímulos externos, poderíamos chegar a conclusão que um tufão é um ser vivo — fogo é um ser vivo, o burro (=jumento+égua) não é ser vivo e formiga operária não é ser vivo. Como já sabemos que um tufão NÃO é um ser vivo, então mudamos nossa definição para que o fenômeno natural seja excluído. Podemos incluir exigência de carboidratos, proteínas, ácidos nucleicos, etc. [1)]. Assim, vamos "definir" sistemas complexos a partir de algumas propriedades comuns. Por sorte, às vezes é mais fácil estudar um sistema do que definí-lo [2)].

Um sistema é um conjunto de elementos interagentes, interconectados que formam o todo. Os elementos podem ser indivíduos, células, órgãos (sistema nervoso), idéias, planetas (sistema solar), regras (ir contra "o sistema"), etc. Para um sistema ser considerado complexo, o mesmo deve apresentar um comportamento que não é ser facilmente previsto a partir do comportamento de seus componentes — o sistema solar não deve ser considerado um sistema complexo, pois o comportamento do sistema é facilmente previsível a partir das informações dos seus componentes e da lei da gravitação. Um sistema complexo deve apresentar um número grande de componentes. Outras propriedades características de sistemas complexos são

- **Não-linearidade:** as interações são não lineares e, em geral, entre vizinhos. Graças ao caráter não-linear, pequenas pertubações podem causar grandes (ou nenhum) efeito. A vizinhança é determinada pelas interações, não sendo havendo necessariamente nenhuma ligação com as distâncias geomé-

tricas entre os elementos. Você pode interagir com alguém de pensamento "próximo" ao seu, não necessariamente seu vizinho (frequentemente os vizinhos não tem gosto musical próximo ao seu e nem gostam de ouvir aquelas músicas no mesmo horário que você).

- **Feedback:** ou retro-alimentação. Os efeitos das ações dos elementos constituintes do sistema determinam como os próprios elementos atuarão no futuro. Os feedbacks podem ser positivos ou negativos, no sentido dos elementos guiarem suas ações da mesma forma, ou de forma contrária, da ação previamente tomada.

- **História** Devido aos feedbacks e as relações não-lineares, a história dinâmica do sistema é importante. Pequenas pertubações podem levar a estados totalmente diversos.

- **Emergência:** comportamentos e padrões macroscópicos (do tamanho do sistema) aparecem a partir das interações locais microscópicas (do tamanho dos componentes).

- **Auto-organização:** o sistema vai para estados especiais sem que seja guiado por um líder, como células que se juntam para formar um órgão que tem alguma função específica. O mercado financeiro é um outro exemplo espetacular de auto-organização: agentes agem segundo as regras do mercado, de forma mais ou menos independente, mantendo o mercado em funcionamento de maneira consideravelmente estável e capaz de resistir a grandes flutuações.

Com as propriedades acima, podemos enumerar vários sistemas simples e complexos:

Sistemas Simples

Gás ideal

Sólidos

Sistema Solar

Teoria Quântica de Campos Relativística, bosonização em $2+1$ dimensões, etc.

Sistemas Complexos

família, torcidas, partidos, governos, sindicatos

clima, ecossistemas

computadores, softwares, Internet

tecidos, órgãos, cérebro, corpo humano

mercado financeiro, firmas, corporações

O estudo de sistemas complexos difere das ciências tradicionais no sentido que nestas últimas o conhecimento tem-se dado na direção de especialização: físicos nucleares de altíssimas energias usam ferramentas e métodos que não se aplicam em reações nucleares perto da barreira coulombiana, ou os de elementos superpesados não se aplicam em íons leves. Em Medicina, é possível se tornar um expert em cirurgia plástica de abdômen ou lipoaspiração. O autor reconhece a necessidade destes especialistas e não quer falar mal deles pois pode precisar dos mesmos futuramente (em particular, os de lipo). Em sistemas complexos, ao contrário, buscamos estudar os comportamentos universais: o que o mercado econômico, fazer um café e perfuração de petróleo têm em comum ? Desta forma, o estudo de sistemas complexos vai na direção da interdisciplinaridade. As ferramentas e os métodos de estudo portanto se aplicam em diversas áreas. Podemos dizer que o estudo de sistemas complexos é um "update" da Alquimia.

Algumas características de sistemas complexos estão presentes em alguns fenômenos físicos: as relações entre os elementos de um sistema podem ser conflitantes, como nos sistemas desordenados. Por exemplo, um dos modelos de redes neurais tem uma analogia bastante forte com o problema físico de vidros de spins (sinapses

excitatórias/inibitórias $\leftrightarrow$ interações ferro/antiferromagnéticas). A estrutura dos sistemas complexos lembra a ausência de comprimento característico em transições de fase. Entretanto , algumas características particulares de sistemas complexos impedem o uso de várias ferramentas tradicionalmente usadas nestes estudos:

- estudar as partes não é suficiente para entender o problema. A extrapolação para um sistema grande não é, assim, trivial.
- a interação com o ambiente toma papel importante na dinâmica do sistema.
- as flutuações são grandes e muitas vezes são as grandezas de interesse. As funções não são suaves, o que dificulta o estudo por equações diferenciais.
- as interações são frequentemente assimétricas.
- nem sempre podemos reduzir a dinâmica para alguns poucos parâmetros relevantes.

Desta forma, simulações computacionais aparecem como uma ferramenta poderosa para este tipo de estudo. Ainda que poderosas, as simulações precisam ser bastante eficientes já que o custo computacional é alto (grande número de elementos e interações, tempos de relaxação enormes), apesar da simplicidade na descrição do problema. Em particular, muitos sistemas complexos podem ser reduzidos à forma binária (verdadeiro ou falso, $\pm 1/2$ no caso de spins, presente ou ausente, alta ou baixa concentração, etc.). Esta simplificação permite uma implementação bastante eficiente em computadores digitais, permitindo a execução de várias operações simultaneamente em um único processador. Por implementação eficiente, entenda-se: velocidade na execução e conomia de memória. Nas seções seguintes veremos algumas técnicas computacionais e aplicações das mesmas em sistemas complexos variados.

2 Operações com Bits

Computadores lidam com dois tipos de números: inteiros e de ponto flutuante. Nós vamos utilizar, sempre que possível, os números inteiros pois as operações são mais rápidas (números de ponto flutuante exigem operações nas mantissas e nos expoentes). É mais fácil trabalhar com números inteiros, em sistemas digitais, na base dois: um número inteiro é uma sequência de zeros e uns – bits (*bi*nary digi*ts*. A sequência de oito bits é chamada byte. Os computadores lidam com sequências de bits, cujo tamanho depende do processador (ou CPU – *central point unit*). A maioria dos microcomputadores hoje (2004) tem processadores de 32 bits. Novos processadores de 64 bits começam a aparecer (Opteron, Itanium e as mais antigas Alphas). O PlayStation II usa um processador de 128 bit, o que justificaria um projeto às agências financiadoras já que o mesmo roda Linux [3)]. A sequência de bits cujo tamanho é igual a capacidade do processador é chamada palavra ("word").

Os estados dos elementos constituintes de um sistema complexo frequentemente podem se reduzidos a uma forma binária: spins ($s_i = \pm 1/2$), presença ou ausência, elementos em uma mistura binária, altas e baixas concentrações de um elemento no sistema imunológico (antígenos, anticorpos, etc.), neurônios emitindo ou não um sinal pelas sinapses, comprar ou venda no mercado financeiro, e muitos outros exemplos. Armazenando os estados em bits ao invés de 4 bytes (como na programação tradicional), a economia de memória é de um fator 32! A programação utilizando bits exige um custo maior na atenção do programador. Sendo a memória RAM (*Random Access Memory*) um dos componentes mais baratos de um computador, hoje em dia, pode-se questionar a validade desta economia. No entanto, a memória mais rápida de um computador é a memória cache (da ordem de 15 vezes mais rápida que a RAM tradicional).O cache funciona "próximo" ao processador e armazena os dados mais utilizados frequentemente. Devido ao alto custo, a memória cache é menor que a RAM (de 128Kb nos Celerons mais antigos passando por 512/1024Kb nos AMD e Pentiuns

mais modernos e 4Mb nas Alphas). Além do ganho de memória existem um ganho ainda maior: operações com bits são realizadas em paralelo (mesmo em um só processador) e não é só isso: programando em bits, você poupa memória, realiza até 64 operações ao mesmo tempo e ainda leva grátis – para os 100 primeiros telefonemas – o fato que cada operação binária é intrinsicamente mais rápida que as operações tradicionais com inteiros. Se você perde muito tempo com suas simulações, o Windows dá pau antes delas terminarem, e ainda tem que salvar em disco grandes quantidades de disco: seus problemas acabaram com a simulação com operações de bits com um "Multi-Bit Simulator Tabajara´´. Se não bastasse toda esta propaganda, simular múltiplas instruções em um só processador [4)] é ainda mais eficiente que um cluster de computadores rodando em paralelo já que a comunicação é realizada dentro do processador, independente da velocidade e confiabilidade da rede. Para aqueles que gostam de programação "hardcore" (ou "*escovar os bits*") vale a pena citar que a programação em bits pode ainda ser otimizada utilizando-se de instruções especiais primariamente desenvolvidas para aplicações multimídia e jogos: instruções MMX e SSE em micros computadores [4)].

Uma das principais razões das quais operações com bits são mais eficientes que a programação tradicional é o fato que operações tradicionais não podem ser paralelizadas, ou seja, realizadas em todos os bits de uma só vez. Em operações como soma, multiplicação, existe o "vai um", ou "carry", onde você precisa saber o resultado da operação no bit anterior (a direita) para processar a operação no bit seguinte. As operações Booleanas [5)] podem ser aplicadas a todos os bits de uma só vez. As operações Booleanas de interesse estão reunidas na tabela 1.

Questões

Q.1 Na tabela 1 mostramos 3 operações binárias. Quantas operações binárias existem no total ?
Q.2 Quantas operações binárias dão resultado zero se ambos os operandos são nulos ?

	AND	OR	XOR
00	0	0	0
01	0	1	1
10	0	1	1
11	1	1	0

Tabela 1: Operações binárias AND,OR e XOR. A primeira coluna representa os dois bits como operandos. Embora tenhamos apresentado a tabela com apenas dois bits, é importante lembrar que em um computador digital, as operações são realizadas em paralelo para n bits, onde n é o tamanho da palavra do computador.

Para melhor entender como os deslocamentos funcionam, vamos lembrar como os números são armazenados na base dois. Como exemplo, vamos considerar números de apenas três bits. Com três bits podemos escrever $2^3 = 8$ números diferentes. Assim podemos escolher duas representações: sem sinal, onde os números variam de $[0, 7]$ ou com sinal, onde os números variam de $[-4, 3]$. Os números negativos devem ser representados de tal forma que as operações de soma, subtração, etc. mantenham a mesma estrutura e obviamente os mesmos resultados quando aplicados em quaisquer representações (com/sem sinal). Note que os bits são contados da direita para a esquerda, assim como os dígitos na base dez (unidades, dezenas, centenas, etc.). O bit mais à direita é contado como o bit 0. Abaixo mostramos a representação dos números de três bits com os números negativos seguindo a notação de complemento de dois ($-Y = 2^B - Y$):

$$\begin{aligned} 0 &= 000_2 \equiv 0 \\ 1 &= 001_2 \equiv 1 \\ 2 &= 010_2 \equiv 2 \\ 3 &= 011_2 \equiv 3 \\ 4 &= 100_2 \equiv -4 \\ 5 &= 101_2 \equiv -3 \end{aligned}$$

$$
\begin{aligned}
6 &= 110_2 \equiv -2 \\
7 &= 111_2 \equiv -1
\end{aligned}
$$

Questão

Q.3 Realize operações de soma e subtração com os números acima, na base dois e compare com os resultados na base dez, para se convencer da equivalência das representações.

Ao tentar somar números cuja soma excede oito, vemos que alguns bits são descartados. Este processo é chamado *overflow* e, em geral, fica a cargo do programador a verificação de overflow em alguam operação para evitar resultados errados, às vezes indesejáveis.

Além destas operações binárias, outras operações com bits são importantes para nosso objetivo: os deslocamentos (ou "shifts") e a operação negação. Os deslocamentos de bits estão para a base 2 assim como a multiplicação por potências de 10 está para a base 10. Deslocar um bit para a direita significa multiplicar o número inteiro por 2 (acrescentamos um bit zero na primeira posição). Deslocar um bit para a esquerda significa dividir o número por 2 (e acrescentamos um bit um na última posição). Multiplicar por uma potência n de dois significa deslocar o número n bits à esquerda.

$$
\begin{aligned}
8 &= 2^3 = 001000_2 \\
16 &= 2^4 = 010000_2
\end{aligned}
$$

As operações de bits estão presentes em qualquer linguagem de programação decente, mesmo as mais antigas. Abaixo listamos as operações binárias em C e FORTRAN

```
FORTRAN                          C

integer*4 a,b                    unsigned int a,b;
```

```
a=1024                          a=1024;
b=1023                          b=1023;
write(*,*) iand(a,b)            printf(``%d\n'',a&b);
write(*,*) ior(a,b)             printf(``%d\n'',a|b);
write(*,*) ieor(a,b)            printf(``%d\n'',a^b);
write(*,*) ishft(a,1)           printf(``%d\n'',a<<1);
write(*,*) ishft(a,-1)          printf(``%d\n'',a>>1);
write(*,*) not(a)               printf(``%d\n'',~a);
```

É fácil notar algumas propriedades da representação de um número em binário, por exemplo, um número ímpar tem o valor 1 na posição mais à direita. A representação de um número $2^n - 1$ também é facilmente reconhecível: é uma sequência de n bits com valor 1:

$$\begin{aligned} 7 &= 2^3 - 1 = 000111_2 \\ 15 &= 2^4 - 1 = 001111_2 \end{aligned}$$

Antes de partirmos para as aplicações, vamos praticar construindo máscaras de bits. Embora o aumento na eficiência do programa em bits esteja no alto grau de paralelismo, muitas vezes estaremos interessados em algum bit específico (ou um grupo deles). Para isto fazemos uso das máscaras. Por exemplo, imagine que você queira descobrir qual o estado do bit 7 de uma palavra. Para isto podemos fazer uso de deslocamentos, trazendo o bit 7 para a posição zero. Resta "limpar" todos os outros bits. Assim, para checar o estado do bit n da palavra a, basta fazer

$$bit = (a >> n)\&1 \tag{1}$$

Como a variável *bit* vai conter apenas o valor zero ou um, não é necessário nenhum `if`, que é uma das operações mais custosas do ponto de vista computacional. Vamos utilizar a notação da linguagem C por ser mais compacta, você pode fazer a tradução

para FORTRAN utilizando a pedra de Rosetta acima, da mesma forma que Champolion em 1822, para outras linguagens tão antigas.

Outras máscaras interessantes são (tente compreendê-las):

$x = x\|(1 << i)$	seta o i^o bit
$x = x\&\tilde{}(1 << i)$	apaga o i^o bit
$x = x\hat{}(1 << i)$	troca o estado apenas do i^o bit

Problema

P.1 Escreva um programa que implemente o shift circular: salve as posições dos bits que seriam descartados nas posições em que seriam acrescentados zeros. Esta função corresponde a implementação de condições periódicas de contorno.

Em outras situações, estaremos interessados no número de bits setados para 1/0. Alguns compiladores já tem implementada uma função própria para isto (BITCOUNT ou POPCOUNT). Uma das maneiras de se contar os bits de uma palavra inclui o seguinte algoritmo: para cada bit, contamos os bits setados para um, usando procedimento (1), que dispensa o uso de `if`'s. A operação terá que ser repetida 32 vezes, por exemplo, dentro de um loop. O procedimento seguinte é mais eficiente pois precisa apenas de n operações, onde n é exatamente o número de bits com valor 1 [6]:

```
n=0;
while(x) {
    x=x&(x-1);
    n=n+1;
}
```

Problema

P.2 Escreva um programa que conte os bits dos números de 0 a 65535, segundo os dois procedimentos descritos acima. Obtenha o tempo de computação e compare-os.

Embora o procedimento acima seja elegante, não é o mais rápido. Velocidade em simulações de sistemas complexos é importante já que os procedimentos são repetidos um número muito grande de vezes. O método rápido e sujo de se contar os bits é o "table-lookup", ou "procura na tabela". Basta, no início do programa contarmos os bits representados por uma fração do número total de bits. Por exemplo, contar os bits de todos os primeiros 256 (2^8) números. Guardamos o conteúdo em uma tabela e faremos uso da mesma quatro vezes, para contar os 32 bits de uma palavra. Traduzindo para C:

```
n = bits8[ x & 255] +
    bits8[ (x>>8)  & 255 ] +
    bits8[ (x>>16) & 255 ] +
    bits8[ (x>>24) ];
```

onde `x` é o número do qual queremos contar os bits com valor 1, `bits8[i]` é o vetor armazenando os números de bits com valor 1 no número `i` e, finalmente, `n` é o número de bits 1 em `x`. Um procedimento tão sujo como este serve para determinarmos o logaritmo na base 2 de um número inteiro – que vem a ser o bit mais à esquerda com valor 1.

3 Geradores de Números Aleatórios

Como primeira aplicação das técnicas anteriores, vamos desenvolver a primeira ferramenta para nossas simulações: um gerador de números aleatórios. Enquanto geradores de números aleatórios (RNG, ou "Random Number Generators") são quasi onipresentes em simulações computacionais, os mesmos não são ferramentas confiáveis [7]. Geradores diferentes passam em testes diferentes e são reprovados em outros. Por questão de espaço, não vamos nos deter em análises mais profundas da qualidade de RNG, mas apenas em sua implementação e fundamentos. Recentes revisões podem ser encontradas na literatura como por exemplo, Gutbrod [8].

Os geradores mais conhecidos, e simples, são os chamados LCG's,ou congruenciais lineares, conhecidos desde 1950. Partindo de algum número inteiro positivo r_0 ("semente"), geramos uma sequência pela regra

$$r_{n+1} = (a\, r_n + c) \,\texttt{mod}\, m \tag{2}$$

onde `a mod b` é o resto da divisão de `a` por `b`, a é um número inteiro grande, c é normalmente escolhido como zero e m é escolhido como um número grande (já que o período será no máximo m). Estes geradores são extremamente rápidos (a operação `mod` é o maior gargalo) e passam em diversos testes. A qualidade do gerador é determinada pela qualidade do multiplicador a. Valores típicos e testados para a são [7, 8]: $16807, 69621, 1103515245$; para 64 bits `a`= 13^{13},44485709377909. Faremos $m = 2^{31} - 1$ por uma razão simples: a operação `mod` é a mais demorada, escolhendo m como o maior inteiro positivo dependente da palavra do computador, podemos dispensar a operação `mod`. O fato, já demonstrado anteriormente, que o processador descarta os bits além da palavra é equivalente a pegar o resto da divisão por 2^{32}!!! Assim, em linguagem C, nosso gerador fica simplesmente

```
r = a * r
```

com `a` assumindo alguns dos valores já citados, por exemplo.

Precisamos agora testar o nosso RNG. O primeiro teste é fácil: vamos verificar se o mesmo gera uma distribuição uniforme de valores. Antes de partir para o teste vamos comentar alguns aspectos deste RNG: note que se iniciarmos o gerador com um número ímpar (vamos considerar `a=16807`) só teremos ímpares na sequência. Não é recomendável utilizar números pares já que o gerador poderia eventualmente gerar zero e este é um ponto fixo da dinâmica. Para verificarmos se o mesmo gera uma distribuição uniforme, vamos restringir os valores a um intervalo menor. Um intervalo interessante seria gerar números no intervalo $[0, 127]$ ao invés de $[0, 100]$. A razão é: para gerarmos no intervalo de $[0, 127]$ precisamos de apenas

sete bits, enquanto no intervalo $[0, 100]$ teríamos que dividir o número `r` gerado pelo maior inteiro e depois multiplicar por 100. Ao escolher quais sete bits devemos utilizar, é fácil ver que é melhor considerar os sete bits mais à esquerda (lembre-se que o bit mais à direita é sempre 1, já que os números gerados são ímpares). Para filtrar os bits mais à esquerda utilizamos os shifts ou deslocamentos. Em resumo, a partir de uma semente ímpar, geramos uma grande quantidade de números aleatórios em um dado intervalo e contamos a ocorrência de cada dos resultados. A distribuição de probabilidade de resultados deve ser uma distribuição uniforme.

Problema

P.3 Encontre o histograma de valores para o gerador LCG com `a=16807`, no intervalo de $[0, 127]$, utilizando as sugestões do texto.

O teste acima está longe de ser definitivo já que lida apenas com histogramas e não como a sequência é gerada dinamicamente. Um teste para verificar correlações entre números consecutivos é o método de Monte Carlo para a integração. O método é mais apropriado para integrais de alta dimensionalidade, embora usaremos um exemplo em apenas duas dimensões para ilustração. Vamos obter o valor de π a partir da área da circunferência. A receita para a implementação do método é a seguinte:

1. considere o primeiro quadrante com $x, y \in [0, 1]$
2. geramos um par de números aleatórios (x_r, y_r) no intervalo $[0, 1]$. Isto corresponde a um ponto no primeiro quadrante.
3. se $y_r < f(x_r)$, onde $f(x) = \sqrt{1 - x^2}$, então o par é aceito. Isto significa que o par (x, y) está "dentro" do quarto de círculo.
4. repete-se o processo um grande número de vezes
5. a área do quarto de círculo será a razão entre o número de pares aceitos e o número de pares gerados (já que a área do

quadrante é 1). Para um número grande de lançamentos podemos esperar que o valor se aproxime de $\pi/4$.

Uma simulação típica está mostrada na figura 1.

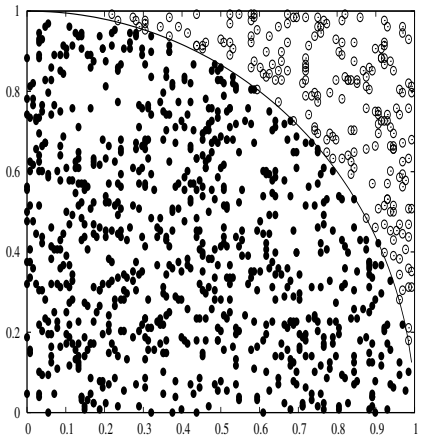

Figura 1: Cálculo de π usando integração por Monte Carlo.

Ainda no exemplo acima, podemos notar que um dado par (x_r, y_r) será aceito com probabilidade y_r. Esta constatação é a idéia principal do método da rejeição de von Neumann para gerar números aleatórios x_r com qualquer outra distribuição $f(x)$ (o método aparece em uma correspondência de von Neumann para S.Ulam, em 1947). O método é muito fácil de implementar e só necessitamos saber o valor da função em cada ponto. Como geramos x_r e y_r de distribuições uniformes, o cálculo de probabilidades (ou valores médios) é feito a partir de médias simples (não ponderadas) e este procedimento é chamado de "amostragem simples"

O método da rejeição pode não ser eficiente se tentarmos gerar números na região onde x_r é muito pequeno (o número de rejeições é grande). A eficiência do método pode ser aumentada se conhecermos alguma função analítica e melhor comportada $g(x)$ que envolva $f(x)$. Ao invés de gerar números no primeiro quadrante, geramos

segundo $x_r \in [0,1]$ e $y_r \in [0, g(x_r)]$. A razão entre o número de aceitos e os pares gerados será proporcional à razão das áreas sob as duas curvas (temos que saber calcular a integral de $g(x)$ para isto) – veja na fig. 2. Este procedimento é chamado de amostragem por importância e é a base do método de Monte Carlo que iremos descrever adiante.

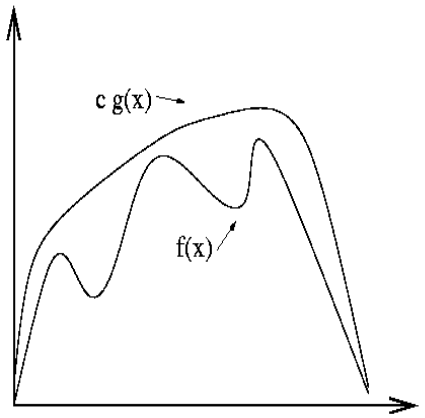

Figura 2: Implementação eficiente do método da rejeição para funções que variam muito rapidamente em função da variável.

Existem muitos outros geradores (esta ainda é uma área de grande debate em Computação) e métodos para gerar números aleatórios segundo diversas distribuições [9]. De particular interesse é a GNU Scientific Library, com vários geradores e aos quais temos acesso aos códigos-fonte (em C) [10].

4 Percolação

O problema de percolação tem aplicações nas mais diversas áreas: física, química, cafezinho, engenharia, prospecção de petróleo, etc.

Porque é um problema simples mas apresenta várias características comuns a vários sistemas complexos, achamos que vale a pena ser apresentado aqui. Considere uma rede quadrada de dimensão $L \times L$. Cada vértice da rede será um sítio. Cada sítio estará ocupado com probabilidade p – concentração de sítios ocupados. No problema de percolação por sítios, a primeira pergunta é: existe algum caminho passando somente por sítios ocupados vizinhos que permita ir da primeira à última linha da rede ? A partir de que concentração de sítios, é garantido que este caminho exista ? Uma variante é considerar não os sítios mas as ligações entre os sítios como presentes ou ausentes: neste caso lidamos com a percolação por ligações. Para maiores detalhes do problema de percolação, existem referências consagradas [11].

Do ponto de vista computacional, o problema consiste em: a partir de uma rede vazia, percorremos todos os sítios e para cada um geramos um número aleatório $\in [0, 1]$ e comparamos com a concentração p. Caso o número gerado seja menor que p, então o sítio é considerado ocupado. Para pequenos valores de p, o que encontramos é que alguns sítios isolados ficam ocupados. À medida que a concentração p aumenta, então aumenta a probabilidade de encontrarmos sítios ocupados vizinhos. A um conjunto de sítios ocupados vizinhos damos o nome de "cluster". O cluster que atravessa toda a rede, para altas concentrações, é chamado "cluster de percolação" (fig.3). Para uma rede 2×2, temos 4 sítios e $2^4 = 16$ configurações possíveis. Uma rede 4×4 tem $2^{16} = 65536$ configurações possíveis. Uma rede 6×6 tem $\approx 10^{10}$ configurações!! Daí o poder do método de Monte Carlo e as simulações computacionais: vamos gerar algums configurações, tantas quantas possíveis e fazer médias nestas configurações.

Algumas quantidades de interesse no estudo do problema de percolação são:

- **limiar de percolação** p_c: concentração a partir da qual aparecem os clusters percolantes. Este valor depende do problema (sítios, ligações, ambos, etc.) e da geometria da rede (quadrada, cúbica, triangular,etc.).

- **probabilidade de percolação** P: probabilidade que um dado sítio pertença ao cluster de percolação. Obviamente esta função só tem valor diferente de zero para concentrações $p > p_c$, onde representa a razão entre o número de sítios no cluster percolante e o número total de sítios.

- **tamanho médio de clusters** $\langle s \rangle$: esta grandeza aumenta com a concentração p, continuamente enquanto $p < p_c$. Para $p > p_c$ não incluímos o cluster percolante, o que faz com que a grandez diminua com à medida que $p \to 1$.

- **distribuição de tamanhos de clusters** n_s: número de clusters com tamanho (ou massa) s, isto é, com s sítios ocupados.

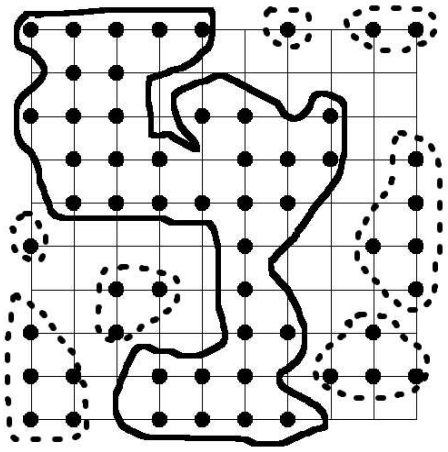

Figura 3: Percolação por sítios na rede quadrada. O cluster circundado pela linha cheia é o cluster de percolação. Note que sítios só são vizinhos se conectados pelas linhas finas (sítios na diagonal não são vizinhos, pertencem a clusters diferentes. Não consideramos aqui condições periódicas de contorno.

Para determinar se uma configuração de rede, gerada com concentração p percolou ou não, fazemos uso do algoritmo de bur-

ning [12]), baseado na propagação de um incêndio em uma floresta. Embora o problema de percolação seja puramente geométrico (após gerada, a rede não muda), vamos introduzir um processo dinâmico no problema:

1. queimamos todos os sítios presentes da primeira linha.
2. sítios vizinhos daqueles que estão queimando, queimarão no passo seguinte
3. sítios queimam por um intervalo de tempo e se tornam queimados, não se queimando novamente.
4. repetimos os passos 2) e 3) até que não exista nenhum sítio queimando.
5. se existe algum sítio queimando na última linha, então a configuração percolou.

Podemos repetir o processo, partindo do sítio da última linha que queimou para identificar o cluster percolante. O número de sítios queimados fornecerá a massa do cluster percolante.

Problema

P.4 Encontre o p_c: para cada $p \in [0.5, 0.7]$, gere 100 configurações iniciais. Implemente o algoritmo de burning. Encontre $P(p)$ para redes $L = 64, 128$ e 256. A concentração crítica p_c será aquela em que as três curvas se encontram. Para a obtenção dos expoentes críticos e uma análise mais detalhada de efeitos de tamnho finito, o leitor deve consultar as referências [6, 11] e em particular o segundo capítulo de Binder e Heermann [13].

O algoritmo de burning serve para identificar a presença de um cluster percolante mas não serve para identificação de todos os

clusters em uma rede. Para isto usaremos o algoritmo de Hoshen-Kopelman [14]. Faremos uso das seguintes variáveis: n_c número de clusters diferentes contados até aquele momento e $idx(i)$ que é o índice do i^o cluster. É mais fácil descrever o algoritmo passo-a-passo na figura (4).

Após a aplicação o algoritmo de Hoshen-Kopelman é possível obter a distribuição de tamanhos de clusters n_s. A distribuição esperada é:

$$\begin{aligned} n_s(p < p_c) &\propto s^{-\theta} e^{-const \cdot s}, \\ n_s(p = p_c) &\propto s^{-\tau} \\ n_s(p > p_c) &\propto s^{-\theta'} e^{-const \cdot s^{1-1/d}} \end{aligned} \tag{3}$$

Note que para $p \neq p_c$, o comportamento para grandes s é governado pela exponencial decrescente. Isto significa que existem tamanhos máximos para cada p. Somente em $p = p_c$ encontramos um comportamento tipo lei-de-potência para os tamanhos de clusters. Isto significa que não existe um tamanho característico de cluster: todos os tamanhos até o cluster percolante (existe apenas um) estão presentes e os menores aparecem em maior número. É como este texto: provavelmente nenhum outro texto sobre sistemas complexos demorou tanto tempo para apresentar uma lei de potência! Uma tentativa de mostrar a distribuição de clusters na rede quadrada é apresentada na figura (5).

Problema

P.5 Implemente o algoritmo de Hoshen-Kopelman. Verifique a validade das equações 3, fazendo gráficos log-log de $n_s \times s$, para $p = 0.55, 0.59, 0.65$. Faça redes tão grandes quanto 1024×1024

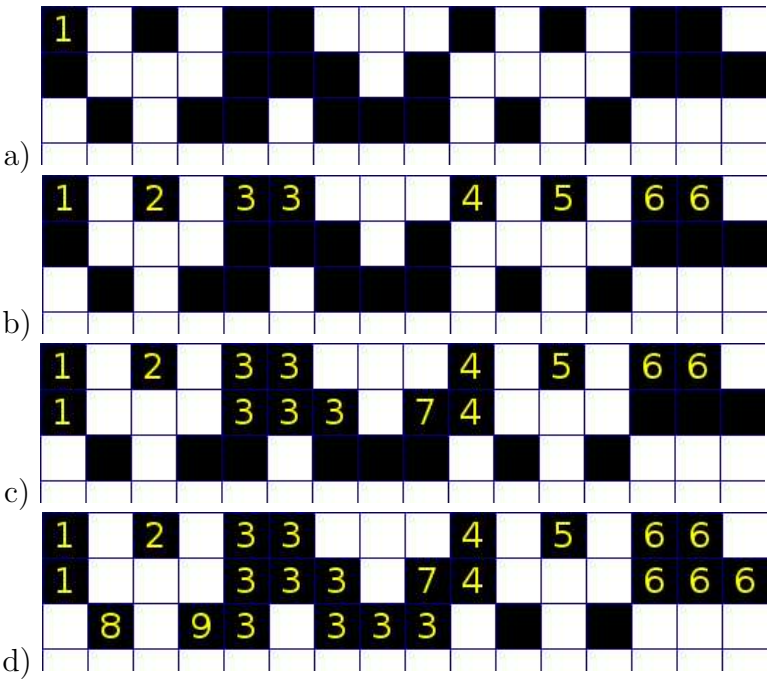

Figura 4: a) Iniciamos contando os clusters da 1^a linha e à esquerda. No primeiro sítio presente identificamos o primeiro cluster e o indexamos como 1 ($n_c = 1$, $idx(1) = 1$). b) Se o sítio testado tem um vizinho que já foi rotulado, este recebe o mesmo rótulo do vizinho à esquerda, como no cluster 3 . ($n_c = 6$, $idx(1) = 1, idx(2) = 2 \ldots idx(6) = 6$). c) Da segunda linha em diante, o sítio procura por vizinhos rotulados à esquerda e embaixo. No caso de conflito entre vizinhos, o sítio assume o número mais baixo e as váriaveis n_c e $idx(i)$ são atualizadas de acordo: no caso $n_c = 6$ porque o cluster sete é o mesmo que o quatro, logo $idx(7) = 4$. d) utilizamos a mesma regra anterior. Atualizamos $idx(9) = 3$ e em sequência $idx(7) = 3 \longrightarrow idx(4) = 3 \longrightarrow idx(3) = 3$. Finalmente após toda a rede varrida, atualizamos todos os sítios seguindo o vetor idx.

Questão

Q. 4. Como poderíamos generalizar o algoritmo de Hoshen-Kopelman para dimensões maiores que 2 ? Seria possível escrever um programa genérico, para d dimensões ?

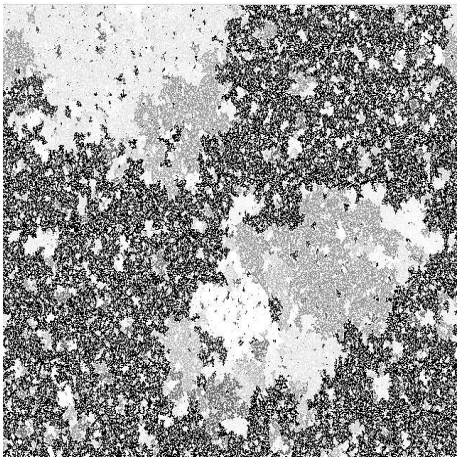

Figura 5: Percolação por sítios na rede quadrada. Distribuição de clusters em p_c. O cluster mais escuro é o cluster de percolação. Outros clusters são mostrados em diferentes tons de cinza: notem a diversidade de tamanhos de clusters.

Agora que temos e dominamos (???) as ferramentas básicas para simulações computacionais vamos partir para o mais fácil: as aplicações. Não será possível descrever em detalhes todas as aplicações apresentadas no mini-curso, portanto optaremos por deixar um roteiro de possíveis experiências computacionais para aqueles que sobreviveram até esta parte. As apresentações que estão disponíveis no site contém um enfoque mais chocante e sensacionalista, repleta de ilustrações.

5 Caminhadas Aleatórias

Gerar caminhadas aleatórias unidimensionais em um computador é uma tarefa fácil. Basta usar o gerador proposto acima e selecionar o bit mais à esquerda. Não é preciso `if`'s para esta tarefa. Se a variável `x` contém a posição do caminhante aleatório, basta fazer

```
r=r*16807
x=x+(2*(r>>31)-1)
```

Em detalhes: `r` é o número aleatório, `(r>>31)` seleciona o último bit (0 ou 1), `(2*(r>>31)-1)` portanto pode assumir os valores ± 1 com igual probabilidade. Com isto geramos caminhadas aleatórias unidimensionais de uma maneira simples e eficiente. Um truque para simular em duas dimensões é usar o procedimento acima para ambas as coordenadas simultaneamente: o caminhante não andará na rede quadrada `x,y`, mas nas diagonais, eliminando a necessidade de `if`'s! O truque acima não pode ser usado se você deseja simular o caminho aleatório com tendência, onde a probabilidade de ir em uma direção é diferente da probabilidade de ir no sentido contrário. Este pode ser um bom teste para o gerador de números aleatórios: verificar se a dispersão cresce com a raiz quadrada do número de passos.

Existem outras regras de geração de caminhadas aleatórias. Uma delas é a caminhada de Lévy, onde o tamanho do passo não é fixo mas segue uma distribuição de probabilidades:

$$p(l > \lambda) \sim \lambda^{-h} \tag{4}$$

com $0 < h < 2$ e caracteriza a caminhada, $p(l > \lambda)$ é a probabilidade de um passo de tamanho l ser maior que λ. Os passos são independentes, a série é estacionária mas a dispersão diverge (note que há uma probabilidade finita de termos passos muito grandes). Um exemplo está na fig.(6). Embora tenha um comportamento bem diferente do caminho aleatório gaussiano, esta regra não inclui efeitos de memória. Caminhadas como estas são encontradas na natureza, principalmente por animais à procura de alimentos [15].

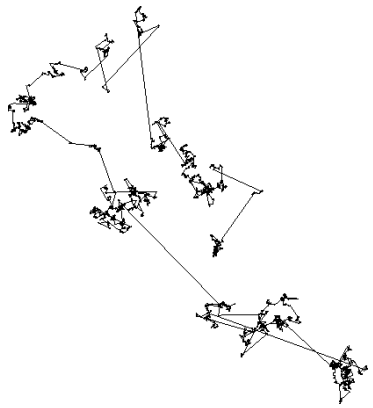

Figura 6: Caminhada de Lévy

Mais interessantes são as caminhadas com efeitos de memória. Um exemplo notável é a caminhada do DNA. O DNA, como sabemos, é formado por uma sequência de nucleotídeos A,T,C e G (adenina, timina, citosina e guanina). Adenina e guanina são chamadas purinas e são compostas de dois anéis, enquanto as purimidinas (citosina e timina) são formadas por um único anel. No DNA, purinas se ligam às pirimidinas para formar um par de bases nitrogenadas (A-T e C-G). Os pares são mantidos por pontes de hidrogênio e formam a base da dupla hélice do DNA. A sequência das bases determina as proteínas a serem sintetisadas. Existem muitas técnicas de identificação das sequências (genes) e com grande percentagem de acertos. Uma técnica interessante, que não reconhece os genes mas permite outras conclusões e especulações, usa uma analogia com os caminhos aleatórios: ao invés de gerarmos números aleatórios, utilizamos as sequências de bases de uma tira do DNA para definir a direção do caminhante. Para várias sequências de genes, encontramos que o caminho aleatório assim gerado é diferente do movimento browniano (fig.7) [16].

O DNA de seres mais evoluídos possui uma quantidade muito grande de genes que não codificam nenhuma proteína (íntrons). Ao

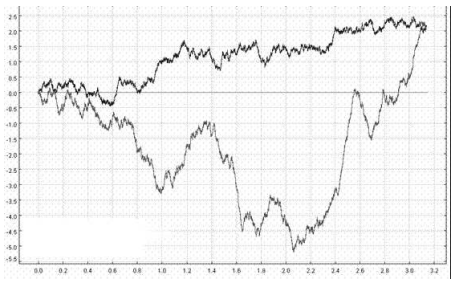

Figura 7: Caminhadas aleatórias construídas a partir de uma sequência de genes. O eixo horizontal corresponde a posição do nucleotídeo na sequência. A linha superior é o caminho aleatório com as mesmas bases porém embaralhadas. A curva inferior corresponde à sequência original do DNA. É notável que a curva inferior tem uma maior tendência em manter a direção do movimento.

tentar construir caminhos aleatórios diferentes para éxons (partes que codificam) e íntrons encontrou-se [16] que a correlação na parte que não codifica é maior que na parte "útil". Isto significa que existe maior redundância nos íntrons. O DNA das espécies evoluídas é bastante fragmentado, com os comprimentos de sequência de íntrons muito maiores que as sequências de éxons (de 10 a 100 vezes maiores). Em vistas destes resultados, especula-se que os íntrons correspondam ao DNA fóssil, isto é, um registro de nossas mutações fig.(8). Um outro sistema bastante diverso mas que apresenta estas características de evolução (inclusão, remoção e mutação) é o armazenamento de arquivos em disquetes. Quando o computador recebe a ordem de apagar um arquivo o que ocorre, pelo menos no DOS/Windows, é apenas a retirada do indicador do arquivo no índice (FAT ou "file allocation table") e subsequente liberação dos setores ocupados pelo arquivo: a informação permanecerá no disco até que outra seja superposta. À medida que um arquivo vai sendo editado e muda seu tamanho, partes antigas também permanecm no disco. Isto vai criando uma área de arquivos que estão salvos no

disco (equivalente aos éxons) e outra que vai sendo deixada como história (íntrons). Os resultados das análises de correlação e flutuação são idênticos aos do DNA, sem que tenhamos que esperar por várias gerações para ser identificado [17)]

Caminhadas aleatórias podem aparecer em outras áreas bastante diversas. Uma das mais antigas é o modelo de Bachelier para o mercado financeiro (1900). Em sua tese, Bachelier sustenta que o mercado é um jogo justo pois a chance de cada participante é igual de sucesso ou fracasso, pois o preço flutuaria como um caminho aleatório. Hoje sabemos que o mercado não se comporta desta maneira. Segundo a hipótese do "mercado eficiente", o mercado reage rapidamente e completamente às novas informações e mostra esta resposta no preço das ações. Verifique na figura 9 as comparações com caminhos aleatórios variados e séries do mercado financeiro.

Além do comportamento dinâmico do mercado, verificado através das séries temporais apresentadas aqui, existem outros estudos relacionados aos temas aqui tratados. Um deles diz respeito às variações nos preços das ações em diferentes intervalos de tempo [18)]. Para intervalos grandes (da ordem de dias e meses), a variação nos preços das ações segue uma gaussiana, como poderíamos esperar, já que é grande o número de fatores envolvidos. Entretanto, para variações da ordem de minutos e poucas horas, o histograma das variações apresenta "fat-tails", ou caudas largas. Ao invés do decaimento exponencial, o histograma decai como lei de potência, portanto eventos extremos acontecem com maior frequência do que apareceriam se fossem puramente aleatórios. Veja os resultados na fig.(10) para o índice S& P 500 (Standard and Poor's das 500 maiores empresas americanas).

Um modelo simples que propõe uma explicação para o aparecimento de caudas largas e que é baseado em idéias de percolação é o proposto por Cont-Bouchaud [19)] e estudado através de simulações de duas a sete dimensões [20)]. Segundo os autores, este comportamento aparece devido à diversidade do tamanho de clusters de agentes que corresponde a um grande número de grupos que to-

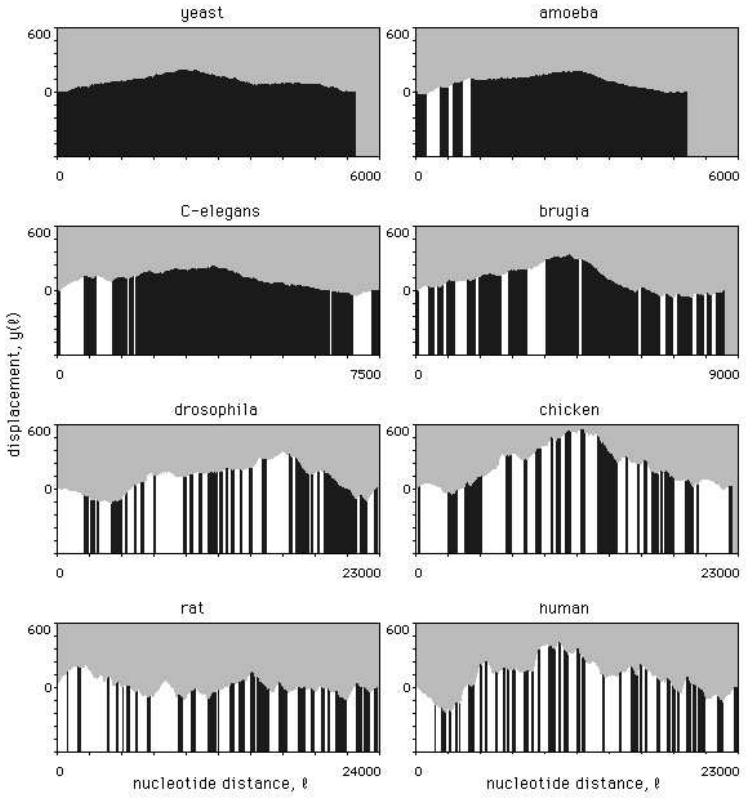

Figura 8: Caminhadas aleatórias construídas a partir de uma sequência de genes MHC (myosin heavy chain) para várias espécies. A parte escura corresponde aos éxons e a parte mais claro aos íntrons. Observe que para espécies mais evoluídas (parte de baixo das figuras) a rugosidade do caminho é mais elevada, devido a presença de correlações nas sequências de íntrons.

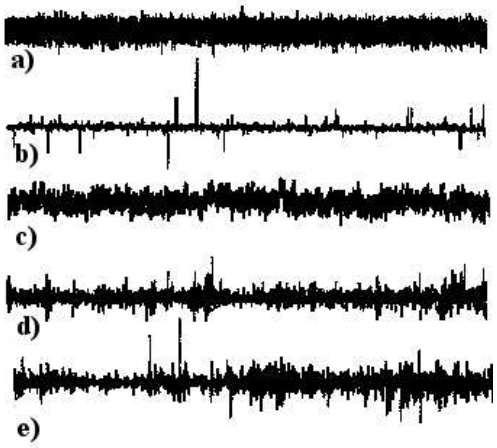

Figura 9: a) Variações em um caminho aleatório sem tendências ou memória.
b) Variações em um caminho de Lévy. Observe os saltos maiores, acontecendo de tempos em tempos.
c) Movimento browniano multifractal: a série não é estacionária.
d) Mudanças no preço das ações da IBM. Os saltos maiores estão dentro de uma área de maior agitação (resposta do mercado)
e) cotação dólar-marco, comportamento global semelhante ao item anterior.

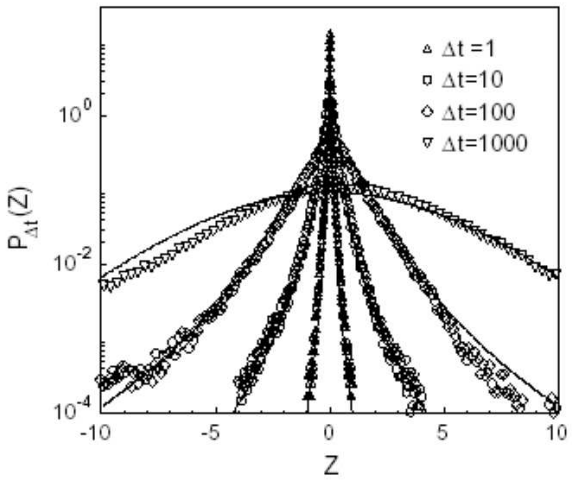

Figura 10: Histograma das variações dos preços das ações para a Bolsa de Nova York (NYSE). Enquanto para grandes períodos de tempo (dias) o histograma é gaussiano, o mesmo sofre um crossover para um comportamento com caudas largas (ou decaimento tipo lei de potência), quando as variações são da ordem de minutos.

mam decisões de modo semelhante mas que têm poder econômico também bastante diverso. Os tamanhos destes grupos também parecem estar distribuídos como leis-de-potência, embora seja difícil confirmar experimentalmente este comportamento.

6 Redes Aleatórias

Recentemente, um problema que tem chamado bastante a atenção dos pesquisadores de sistemas complexos é a arquitetura de relações que suportam estes sistemas. Até agora, nestas notas, vínhamos tratando de geometrias simples como caminhos unidimensionais ou redes regulares. No entanto, uma fração considerável dos sistemas complexos não está disposto ou conectado através de redes regulares. A organização dos neurônios no cérebro é um belo exemplo: em algumas regiões (como o cerebelo) a conectividade é grande, em outras regiões parecem estar dispostos em camadas enquanto em outras regiões a conectividade é baixa. O grau de conectividade determina a função (ou vice-versa).

Redes aparecem nas mais diversas situações, para deleite dos complexistas: neurônios no cérebro, Internet, redes de terroristas, redes elétricas, espécies de um ecossistema, palavras em um artigo ou livro, sistemas de transportes, reações bioquímicas no corpo humano, etc. Em geral estas redes não estão conectando componentes de maneira aleatória ou desordenada: parece existir uma direção preferencial de conexão. Redes "scale-free", ou livre de escalas, são aquelas em que a distribuição de conexões por sítios segue uma lei-de-potência: temos poucos sítios (ou elementos) com altíssimo grau de conectividade e muitos sítios com conectividade baixa. A Internet, a Web, sistemas viários, parecem se comportar como redes scale-free.

Antes de mostrar outras aplicações, vamos brevemente descrever como construir uma rede deste tipo. Confira a receita no caption da fig. (11). Note que, por acaso, alguns sítios se conectam mais que os outros. Outros que começam mais conectados pordem perder sua vantagem enquantos outros consolidam seus privilégios. Sítios

que são adicionados mais recentemente têm menor probabilidade de serem os mais conectados. Um fator importante e que determina a distribuição de conexões neste modelo é o número de conexões adicionadas por cada novo indivíduo. Uma comparação entre redes scale-free e grafos aleatórios está apresentada na fig.(12).

Uma característica peculiar das redes scale-free é que as mesmas são robustas frente a remoção de elementos: cerca de 80% da Internet teria que ser destruída para causar a interrupção da mesma. Por outro lado, a rede é extremamente susceptível a ataques localizados: se os hubs mais conectados são os destruídos, invadidos ou contaminados, o desastre é total (não se preocupem tanto, estes rodam unix). Um exemplo, quase ninguém fica sabendo quando a rede "cai" em Niterói (eu sei!!!), mas se o sistema do CBPF é o afetado, o dano é muito pior. Outra característica importante é que as distâncias nestas redes são muito diminuídas: por exemplo, em média levamos 19 cliques para ir de uma página a qualquer outra página na Web, embora o número de páginas seja extremamente alto (da ordem de bilhões). Isto também facilita a propagação de epidemias em populações. Estas propriedades já estão sendo colocadas em prática: veja um exemplo no Google – fig.(13). A procura por "linux"retorna mais de 90.000.000 de páginas, no entanto, as mais procuradas aparecem primeiro (linux.com, redhat.com, suse.com, mandrake.com) mesmo que não tenham linux no endereço, apenas no conteúdo.

Outra geometria de redes aleatórias que também causa interesse é a rede de mundo pequeno ou "small world" [22]. Nesta geometria, começamos com uma rede regular e aleatoriamente selecionamos dois sítios que receberam uma ligação que encurtará a distância entre eles. Estas redes diferem das redes scale-free por apresentarem um alto coeficiente de agregação (clustering), isto é, o número de vizinhos dos vizinhos que também são vizinhos diretos de um dado sítio (este coeficiente é zero na rede quadrada, por exemplo). Veja a fig.smallworld. Este parece ser uma modelo mais adequado para descrever epidemias que as redes scale-free. Uma aplicação curiosa é o oráculo de Bacon (Kevin Bacon, ator): de um

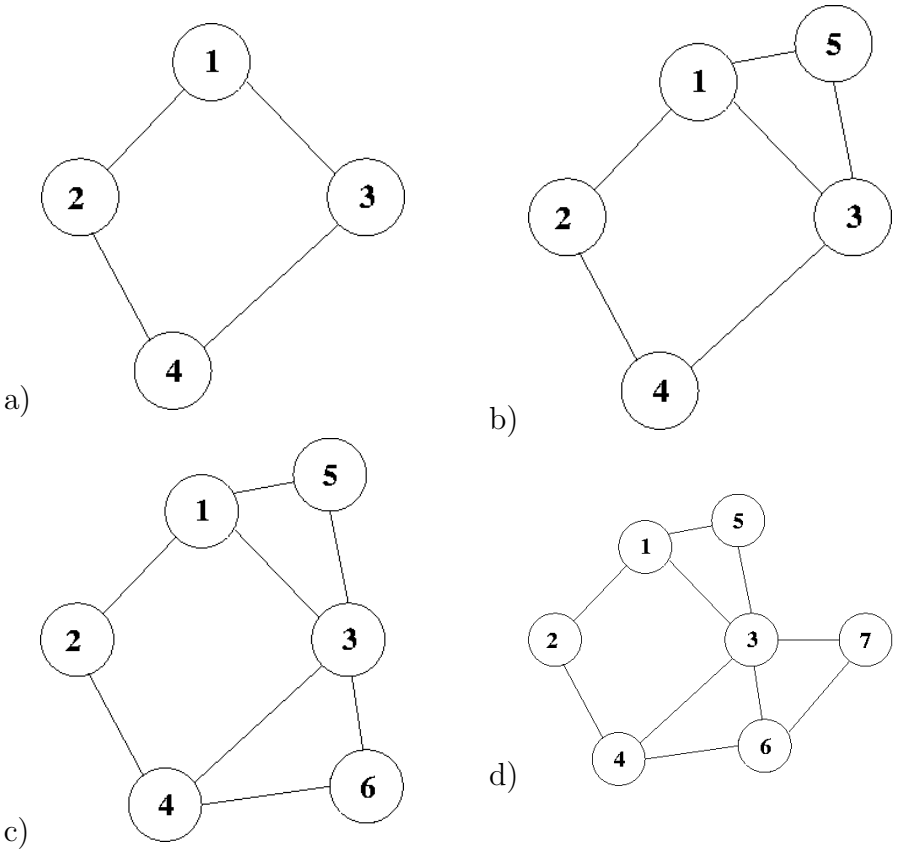

Figura 11: Etapas de construção de uma rede scale-free
a) começamos com alguns pouco elementos com duas conexões cada
b) um novo elemento é adicionado. Cada elemento entra com duas ligações. Neste caso, todos os elementos existentes tinham a mesma probabilidade de receber as ligações. Após a introdução do novo elemento, os sítios 1 e 3 têm maior probabilidade de receberem novas conexões.
c) O novo elemento escolhe novamente dois outros para se conectar. O sítio 3 já tinha maior probabilidade que os outros, graças ao sorteio do 6, esta superioridade é aumentada. O sítio 4 agora tem a mesma probabilidade de receber uma nova conexão que o sítio 1.
d) O sítio 7 escolhe dois vizinhos e o 3 é novamente sorteado.
O truque computacional bastante simples aqui é manter um vetor com todos os sítios que recebem ligações (considere cada ligação como tendo dois sítios recebendo as ligações). Ao sortear aleatoriamente qualquer elemento do vetor, aqueles que tiverem um maior número de ligações será escolhido com maior probabilidade.

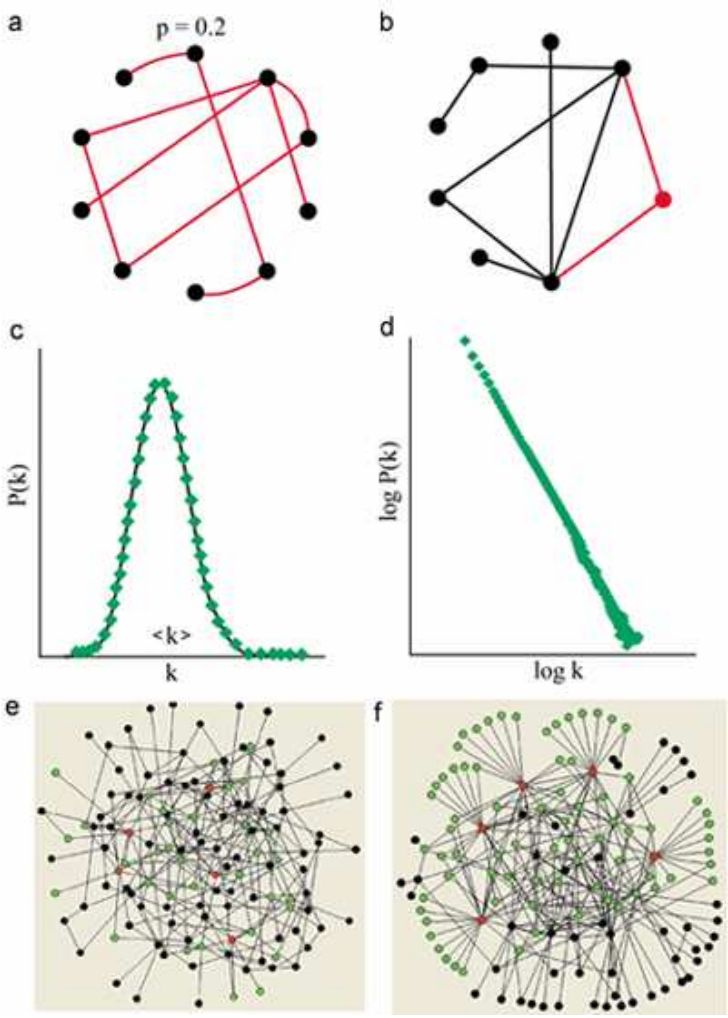

Figura 12: a) Rede aleatória b) Rede Scale-free
Histograma dos números de ligações por nó para c) rede aleatória
d) rede scale-free
redes maiores e) aleatória f) scale-free

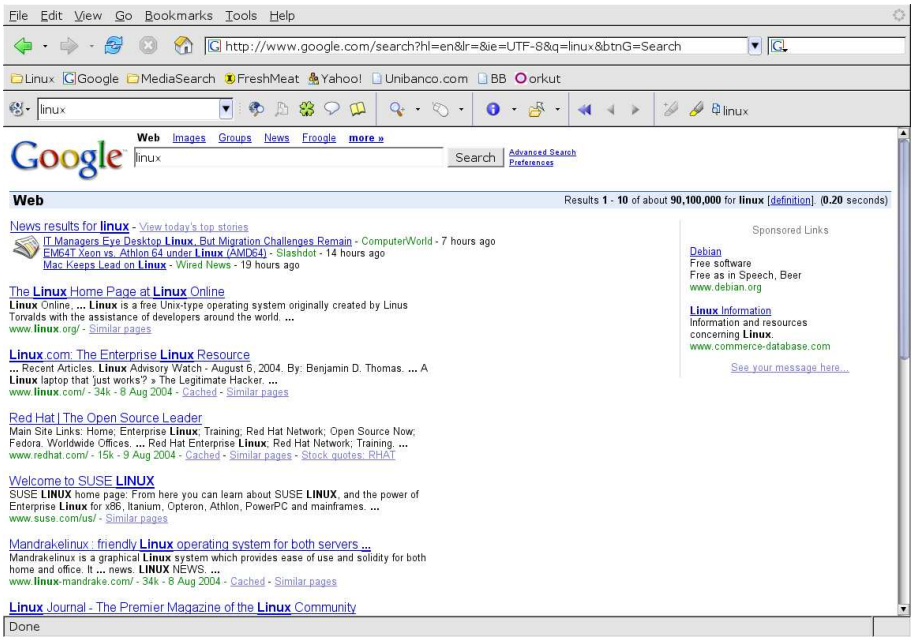

Figura 13: Procura no Google por “linux”. Apesar das mais de 90.000.000 de páginas, aquelas com maior número de links referenciando à elas, aparecem primeiro.

banco de dados de atores e atrizes que contracenaram em filmes americanos, é obtida a "distância" (número de atores que contracenaram no mesmo filme) a Kevin Bacon. Por exemplo, Fernanda Montenegro contracenou com Jeanne Moureau em Joanna Francesa (1973), que contracenou com Eli Wallach em The Victors (1963) que contracenou com (adivinhem??) Kevin Bacon em Mystic River (2003).

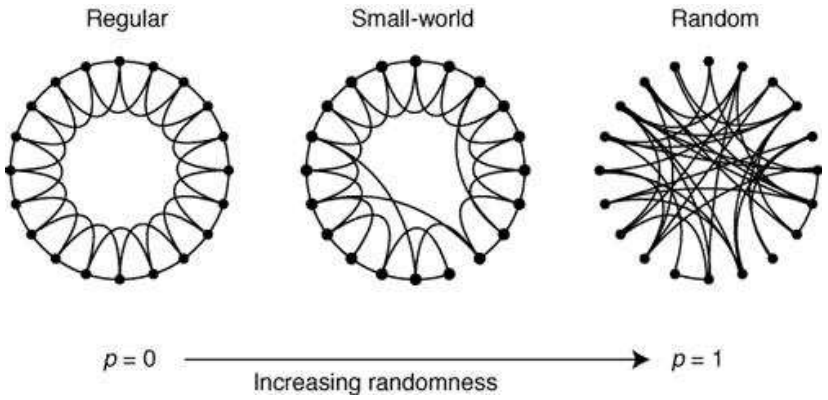

Figura 14: Construção de uma rede small-world. Partindo de uma rede regular, com probabilidade p, transformamos ligações de longa distância em uma ligação local (a presença de uma ligação faz a distância entre os sítios cair a um). No limite $p \to 1$ temos a rede aleatória. Para pequenos valores de p, a distância entre dois sítios cresce somente com o logarítmo do número de sítios, como nas redes aleatórias, porém a agregação é alta, como nas redes regulares.

7 Conclusões

Procuramos introduzir aqui, algumas das ferramentas e conceitos no estudo de sistemas complexos. Este material não cobre todo o material apresentado no curso, mas deveria servir como o pontapé

inicial e referência na implementação das simulações computacionais. Na home-page do autor, o leitor poderá encontrar as notas de aulas, além de outras apresentações no mesmo tema.

Finalmente, gostaria de agradecer o convite, o carinho e a atenção dos responsáveis pela Escola, antes, durante e depois da conclusão da mesma.

Referências

1. Francisco J. Varela, Humberto R. Maturana, and R. Uribe, Biosystems, **5**, 187 (1974).

2. Linus Pauling, The Origin of Life on Earth; ed: A. I. Oparin, (ed. New York: MacMillan, 1938).

3. Linux for PlayStation 2, `http://playstation2-linux.com/`

4. Linux Parallel Processing HOWTO
`http://www.tldp.org/HOWTO/Parallel-Processing-HOWTO.html`

5. George Boole, The Mathematical Analysis of Logic (1847)
George Boole. "An Investigation of the Laws of Thought" in Classics of Mathematics, Ronald Calinger, pg 557-565, (ed. New Jersey: Prentice Hall, 1995).

6. P.M.C. de Oliveira, Computing Boolean Statistical Models, ed. World Scientific (1991)

7. S. K. Park and K. W. Miller, "Random number generators: good ones are hard to find", Communications of the ACM 31 (1988) 1192-1201.

8. F.Gutbrod, "New Trends in Pseudo-Random Number Generation" in Annual Reviews of Computational Physics IV, ed. D. Stauffer, 203-257 (1999).

9. W.Press *et al*, "Numerical recipes in C", (ed. Cambridge University Press; 2nd ed. 1992)

10. The GNU Scientific Library
`http://www.gnu.org/software/gsl`

11. D. Stauffer and A. Aharony, Introduction to Percolation Theory, (ed. Taylor and Francis, London 1994)

12. H.J. Herrmann, D.C. Hong and H.E. Stanley, J.Phys. A17, L621 (1984).

13. K. Binder and D.W. Heermann, "Monte Carlo Simulation in Statistical Physics", (ed. Springer-Verlag 1997)

14. J. Hoshen and R. Kopelman., Phys. Rev. B. 14, 3438–3445 (1976).

15. G.M. Viswanathan *et al*, Nature 381, 413 (1996).

16. C.K. Peng *et al*, Nature, 356: 168 (1992).

17. G. F. Zebende, P.M.C. de Oliveira and T.J.P.Penna, Phys. Rev. E 57, 3311 (1998).

18. R.N. Mantegna and H.E. Stanley, Nature 383, 587 (1996).

19. J.-P. Bouchaud and R. Cont, European Physical Journal B 6, 543 (1998)

20. D. Stauffer and T.J.P. Penna, Physica A 256, 284 (1998).

21. Albert-Laszlo Barabasi and Reka Albert. Science 286, 509 (1999).

22. D. J. Watts and S. H. Strogatz. Nature, 393 (1998).