Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Luiz Gustavo de Andrade Alves

SIMULAÇÃO COMPUTACIONAL DE SISTEMAS COMPLEXOS

Maringá
2012

Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Luiz Gustavo de Andrade Alves

SIMULAÇÃO COMPUTACIONAL DE SISTEMAS COMPLEXOS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de bacharel em Física, da Universidade Estadual de Maringá.

Orientador:
Prof. Dr. Renio dos Santos Mendes

Maringá
2012

Agradecimentos

Primeiramente, gostaria de agradecer ao professor Renio dos Santos Mendes, tanto pela orientação deste trabalho, como por ter me acompanhado e me incentivado durante toda a graduação, seja nas aulas, nos trabalhos de iniciação científica ou em uma boa conversa.

Também gostaria de agradecer ao Haroldo Valentin Ribeiro, pela acessoria computacional nos trabalhos de iniciação científica e por me ajudar nos primeiros passos na programação e simulação computacional.

Agradeço a minha família, em especial a meus pais, Luiz Carlos Alves e Mirian Cristina de Andrade Alves, e a meu irmão, Cássio Roberto de Andrade Alves, que sempre apoiaram minhas escolhas.

Por fim, agradeço a todos (grupo de Física Estatística e Sistemas Complexos, colegas de sala, professores e amigos) que contribuiram, direta ou indiretamente, na elaboração deste trabalho.

Resumo

Atualmente existe um grande interesse da comunidade científica na compreensão dos padrões e comportamentos de sistemas complexos, visto que eles ocorrem em uma variedade de contextos naturais, da física a ciências sociais. Considerando isso, este trabalho tem como objetivo dar uma visão geral sobre sistemas complexos com ênfase em simulação computacional. Para tanto, faremos as seguintes etapas: inicialmente vamos introduzir brevemente aspectos gerais de sistemas complexos e probabilísticos; em seguida, daremos enfoque aos modelos que descrevem, ao menos aproximadamente, sistemas complexos naturais; também discutiremos temas como caminhada aleatória, caminhada autoexcludente, acreção, fenômenos de espalhamento, percolação de aglomerados, o modelo de Ising e um modelo sobre evolução Darwiniana; por fim, apresentaremos, para cada modelo, alguns resultados obtidos por meio de simulações.

Palavras-chave: Sistemas complexos. Física estatística. Modelos. Auto-organização. Simulação computacional.

Abstract

Nowadays, there is a great interest from scientific society in understanding patterns and behaviours of complex systems, since they occur in a variety of natural contexts, from physics to social sciences. Considering that, this work aims to present an overview about complex systems with emphasis on computer simulation. To do so, we will follow these steps: first we briefly introduce general aspects of complex and probabilistic systems; then we will focus on models that describe, at least approximately, natural complex systems; we will also discuss topics such as random walk, self-avoiding walk, accretion, spreading phenomena, percolation clustering, the Ising model and a model about Darwinian evolution; finally, we will present, for each model, some results obtained through simulations.

Key-words: Complex systems. Statistical physics. Models. Self-organization. Computer simulation.

Sumário

Capítulo 1

Introdução

A investigação científica tenta compreender como a natureza funciona. O entendimento dos sistemas naturais pode ser feito por dois caminhos: pela observação experimental, que, a partir de dados empíricos, nos levam a equações que regem o fenômeno estudado; ou pela modelagem teórica, que consiste basicamente em pensar no problema, em escrever equações e, então, em resolvê-las de forma analítica ou numérica. Sem dúvidas, o uso de equações para descrever a natureza foi e ainda é de grande importância para a compreensão do mundo em que vivemos, pois, com elas, foi possível ter uma primeira aproximação do que ocorre em muitos fenômenos. No entanto, apesar de as ideias resultantes das equações serem de grande ajuda no avanço da ciência, elas possuem algumas limitações, tais como as equações sem soluções ou os sistemas que não podem ser representados em termos de equações. Assim, na busca de modelar processos naturais, outros métodos foram desenvolvidos.

Com o avanço das técnicas computacionais, foi possível desenvolver uma abordagem diferente para entender alguns sistemas naturais, a *física dos algoritmos*[1]. Essa abordagem substitui equações por algoritmos e por programas de computadores. Os programas, quando executados, são usualmente chamados de simulação computacional e modelam diretamente o fenômeno investigado. Em particular, a simulação computacional é uma ferramenta muito útil nos estudos de sistemas complexos.

Apesar de não haver uma definição bem estabelecida sobre o termo complexo, sistemas que são compostos por muitas partes interagindo entre si e que apresentam um comportamento emergente são, em geral, chamados de sistemas complexos. A existência de propriedades emergentes é a única característica mais distinguível de sistemas complexos [2]. Exemplos de tais sistemas frequentemente citados são o comportamento do mercado financeiro, cadeia de moléculas, disseminação de epidemias, redes neurais, evolução biológica, sistema presa-predador e fluxo de automóveis em uma rodovia. Note que tais sistemas estão presentes nos mais diversos contextos, da física dos sistemas complexos a ciências sociais e biológicas, o que justifica o grande interesse em modelos que descrevem sistemas complexos naturais.

Quando não entendemos muito bem o que ocorre em um determinado fenômeno que parece não ter uma ordem lógica, uma primeira aproximação é considerarmos ele totalmente aleatório. De fato, muitos sistemas naturais podem ser descritos pelo modelo de caminhada aleatória. Nesse sentido, cabe ressaltar alguns aspectos relevantes, tais como valores médios, expoentes critícos etc.

A caminhada autoexcludente é uma modificação desse modelo, o qual tem sido muito usado em física dos polímeros. Nessa caminhada, o que difere essencialmente de uma caminhada aleatória é que cada passo dado possui lembranças do passo anterior, de forma que um mesmo lugar nunca é visitado mais de uma vez.

Outros tipos de fenômenos, além daqueles aproximados pelos modelos citados, são de notória importância no campo dos sistemas complexos, como a acreção que, por sua vez, é de grande relevância quando o assunto é agregação ou deposição de partículas. O processo de acreção consiste em partículas movendo-se aleatoriamente até a colisão, momento em que elas se juntam e formam um aglomerado que cresce com o número de colisões.

Ainda em modelagem, os fenômenos de espalhamento também são de grande interesse no entendimento dos sistemas complexos. Por isso, discutiremos o modelo de *Eden*, que descreve uma variedade de processos naturais, tais como o fluxo de fluido através de um meio poroso, a disseminação de epidemia e de rumores.

Como já dissemos, sistemas complexos fazem parte das mais diversas áreas da ciência. A percolação, por exemplo, está presente em biologia (na reação imunológica anticorpo-antigênio), em química (em uma reação de polimerização) e em física (fenômenos críticos). É por isso que devemos dedicar uma parte deste trabalho em um modelo sobre percolação de aglomerados.

Nesse contexto, multidisciplinar e interdisciplinar, temos, ainda, o modelo de Ising. Ele imita o comportamento de sistemas físicos e biológicos tais como ferromagnetismo, redes neurais e disseminação de doenças. Esse cenário ilustra o quanto a modelagem e simulação de sistemas naturais são de interesse das mais diversas áreas do conhecimento.

Muitos sistemas biológicos podem ser explorados por ferramentas da física estatística. Um assunto que permeia sistemas complexos e que deve ser citado é a teoria de Darwin sobre a evolução das espécies. Neste trabalho, daremos enfoque a um recente modelo de evolução que mostra a ocorrência de "coevoluções".

Embora sabemos que existe uma enorme quantidade de informações sobre os temas propostos aqui e que cada um deles poderia ser explorado de forma mais detalhada, iremos nos restringir a aspectos gerais, visto que nosso objetivo é apresentar vários modelos que ajudam no entendimento de sistemas complexos. Para tanto, discutiremos questões sobre os algoritmos e suas aplicações. Os código fonte dos programas aqui utilizados podem ser encontrados no apêndice B, junto a uma breve explicação sobre a função de cada um.

Capítulo 2

Caminhada Aleatória

Imagine que uma pessoa dê um passo em uma direção qualquer, isto é, uma direção escolhida aleatoriamente, e, em seguida, repita o mesmo processo de passos aleatórios até um total de n passos. Esse processo é conhecido como caminhada aleatória (*random walk*) e tem sido muito utilizado por cientistas no estudo de processos estocásticos (probabilísticos).

Em biologia, em particular, esse modelo é muito utilizado, por exemplo, o botânico Robert Brown, em 1828, estudou o movimento de partículas de pólen das plantas [3]. Os físicos, por sua vez, utilizam a caminhada aleatória na descrição de processos com muitas partículas em problemas de termodinâmica e mecânica estatística, tendo como marco inicial um artigo de Albert Einsten (1905), "sobre o movimento de partículas suspensas em um líquido requerido pela teoria cinética-molecular do calor"[4]. Já em economia, esse tipo de movimento é muito utilizado no estudo de séries temporais do mercado financeiro como pode ser visto em [5, 6], por exemplo.

A descrição e a implementação simples torna o modelo interessante do ponto de vista de um estudo computacional. Dessa maneira, abordaremos neste capítulo a construção de uma caminhada aleatória e algumas de suas características importantes.

2.1 Caminhada aleatória em uma dimensão

O modelo mais simples de caminhada aleatória é aquele no qual n passos de tamanhos iguais são tomados ao longo de uma linha horizontal. Assim, tomando como partida o ponto zero no instante inicial, os passos aleatórios podem ser tomados na direção positiva $(+1)$ ou na direção negativa (-1). Então, podemos construir uma lista com os valores aleatoriamente selecionados para os n passos tomados da caminhada unidimensional. Tais valores são gerados por um algoritmo prescrito que cria números aleatórios (veja o apêndice A). Por exemplo, para $n = 15$, podemos ter a seguinte lista de valores:

```
{-1, 1, -1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, 1}.
```

Como escolhemos o zero sendo a posição inicial no instante t_0, se somarmos a cada passo o valor sorteado (+1 ou -1), podemos construir uma tabela com a posição a cada instante t_n. Assim, para a lista anterior, temos:

```
{0, -1, 0, -1, 0, 1, 0, 1, 0, -1, -2, -1, 0, -1, -2, -1}.
```

Seguindo esse raciocínio, construímos o programa *Walk1D*, que gera listas de números aleatórios que formam caminhadas aleatórias em uma dimensão com n passos.

2.2 Caminhada aleatória em duas dimensões

Em duas dimensões, a modelagem é um pouco mais complicada que a anterior, pois agora cada passo pode ser tomado em várias direções. No entanto, vamos considerar uma caminhada aleatória em uma rede quadrada bidimensional. Este tipo de caminhada é normalmente referido como *lattice walk*[1]. Em vez de escolhermos passos em qualquer direção do plano xy, vamos nos restringir a apenas quatro direções: norte, sul, leste e oeste.

Para a construção dessa caminhada, cada passo será representado por um par ordenado, com os possíveis valores:

```
{{0, 1}, {1, 0}, {0, -1}, {-1,0}}.
```

De forma similar à caminhada em uma dimensão, criaremos uma lista com n valores que representam a direção do n-ésimo passo tomado. A seguir, temos um exemplo com $n = 6$:

```
{{0, 1}, {1, 0}, {1, 0}, {-1, 0}, {-1, 0}, {1, 0}}.
```

Dessa maneira, tomando a posição inicial no instante t_0 como sendo a origem e somando o par ordenado escolhido aleatoriamente ao valor da posição anterior, criamos uma lista com a posição para cada instante t_n. Por exemplo, por meio do programa *Walk2D*, geramos a seguinte tabela das posições para $n = 6$:

```
{{0, 0}, {0, 1}, {1, 1}, {2, 1}, {1, 1}, {0, 1}, {1, 1}}.
```

2.3 Análise numérica da caminhada em uma rede bidimensional

Ao estudarmos fenômenos da natureza que envolvem processos aleatórios, características importantes sobre os sistemas podem ser obtidas a partir dos valores médios. Dessa maneira, para análise da caminhada aleatória, vamos calcular dois valores médios: a média do quadrado da distância de ponta a ponta e a média do quadrado do raio de giração. Após isso, dicutiremos algumas relações.

2.3.1 Média do quadrado da distância de ponta a ponta

A distância de ponta a ponta ao quadrado, r^2,de uma caminhada em uma rede bidimensional é dada por:

$$r^2 = (x_f - x_i)^2 + (y_f - y_i)^2, \tag{2.1}$$

[1] *Lattice walk*, termo em inglês que significa caminhada em uma rede.

em que os subíndices f e i representam a localização final e inicial respectivamente. Por simplicidade, escolhendo a origem da caminhada no ponto $\{0,\, 0\}$, temos que:

$$r^2 = x_f^2 + y_f^2\,. \tag{2.2}$$

Para calcularmos o valor médio $< r^2 >$, criamos m réplicas de caminhadas com n passos, somamos todos os r_m^2 e dividimos por m. Por exemplo, criamos uma caminhada de $n = 10$ passos, em que foram feitas 25 réplicas, e obtivemos, por meio do programa *MeanSquareDistance*, o seguinte resultado:

$$< r^2 >= 9,44. \tag{2.3}$$

2.3.2 Média do quadrado do raio de giração

A média do quadrado do raio de giração, $< R_g^2 >$, de uma caminhada aleatória é a soma do quadrado das distâncias de cada passo em relação ao centro de massa dividido pelo número de passos. O centro de massa é a soma das localizações de cada passo dividido pelo número de passos. Assim, temos que a média do quadrado do raio de giração é:

$$< R_g^2 >= \frac{\sum_{m=1}^{n}(r_m - r_{cm})^2}{n}, \tag{2.4}$$

em que $r_{cm} = \sum_{m=1}^{n} r_m/n$.

Para calcularmos o valor médio do quadrado do raio de giração usamos o programa *MeanSquareRadiusGyration*. Com ele, obtivemos, para 25 réplicas de uma caminhada de 10 passos, o seguinte valor:

$$< R_g^2 >= 4,23. \tag{2.5}$$

O raio de giração é uma medida muito usada na física dos polímeros para caracterizar o tamanho, o formato e a anisotropia de uma rede polimérica [7].

2.3.3 Expoente crítico de uma caminhada aleatória

Foi verificado experimental e teoricamente que o valor médio de $< r^2 >$ possui uma dependência do tipo lei de pontência em relação ao número de passos feitos na caminhada. Nessa relação do tipo lei de potência, o expoente é muitas vezes conhecido como *expoente crítico* da caminhada aleatória e pode ser escrito como:

$$< r^2 >= A\, n^{\nu}, \tag{2.6}$$

em que ν é o expoente crítico da caminhada e A é um fator constante.

Em nossas simulações, geramos uma lista de pares ordenados $\{n, < r^2 >\}$ com n de 10 a 200 e incremento de 10 em 10, com 100 réplicas em cada caso. Em seguida, tomamos o logaritmo dos pares ordenados e calculamos o coeficiente de inclinação ν da reta $\log(< r^2 >) = \nu\, \log(n) + \log(A)$, que é o expoente da lei de potência $< r^2 >= A\, n^{\nu}$. Com este procedimento, encontramos um valor para ν de aproximadamente 1 (o valor exato). Na figura 2.1, temos um gráfico obtido a partir dos resultados de nossas simulações.

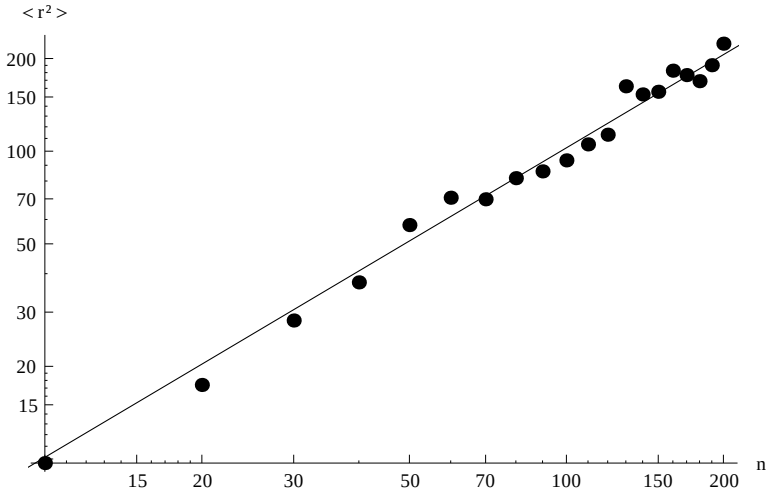

Figura 2.1: Reta obtida por meio de um ajuste linear do logaritmo dos dados simulados, resultando em $\nu \approx 1,005$. Os pontos são os pares ordenados em escala log-log para valores de n de 10 a 200 e incrementos de 10 em 10, com 100 caminhadas para cada caso.

2.3.4 Visualizando uma caminhada aleatória sobre uma rede bidimensional

Um modo interessante de olhar para uma caminhada aleatória é na forma gráfica. Por isso, através do programa *ShowWalk2D*, pegamos a lista gerada por *Walk2D*, traçamos linhas que ligam os sucessivos pontos entre os passos de uma caminhada e criamos uma imagem que representa a nossa caminhada (figura 2.2).

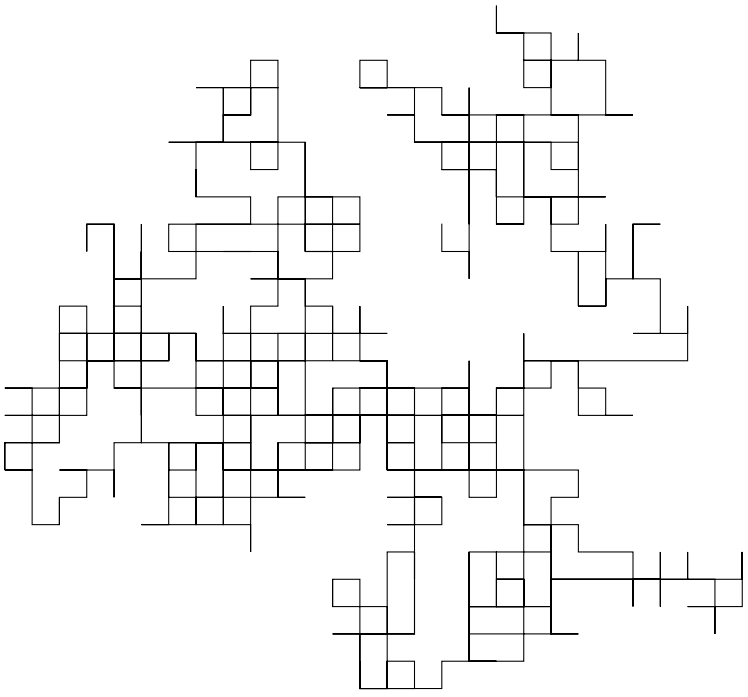

Figura 2.2: Representação gráfica de uma caminhada aleatória com $n = 1000$ passos.

Capítulo 3

Caminhada autoexcludente

Caminhadas autoexcludentes são trajetórias discretas sem autointerseções [8]. Imagine que, em uma caminhada aleatória, o caminhante evite visitar um mesmo local mais que uma vez, talvez porque ele tenha implantado minas terrestres na localização de cada passo tomado, por exemplo. Dessa forma, as propriedades médias desse tipo de caminhada são fundamentalmente diferentes de uma caminhada aleatória.

As aplicações desse modelo ficaram bem conhecidas nas investigações da física de polímeros, veja, por exemplo, a referência [12], na qual podemos encontrar uma discussão mais rigorosa sobre o assunto. As longas cadeias poliméricas, que consistem em várias moléculas conectadas por ligações químicas, podem ser representadas por caminhadas autoexcludentes que abreviaremos por SAW[1]. Outros sistemas que podem ser descritos usando o modelo SAW incluem fenômenos críticos e uma generalização do modelo de Ising para o ferromagnetismo [1]. Um estudo analítico do modelo SAW oferece algumas complicações. Por isso, é de grande valia um estudo computacional para a investigação das propriedades e dos valores médios de variáveis do modelo.

Uma forma de se obter SAWs para o estudo dos valores médios seria, como feito anteriormente para caminhada aleatória, construir várias réplicas e, então, obter tais valores. No entanto, para uma caminhada aleatória, podemos passar por um mesmo local mais de uma vez, enquanto que, em uma SAW, isso não é possível. Por esse fato, construir uma caminhada autoexcludente por esse método é inviável, pois a simples ocorrência de um passo errado, ou seja, um passo para um local já visitado, torna necessário descartarmos a caminhada e recomeçarmos o processo de construção. Além disso, quanto maior a caminhada, maior a probabilidade de ter um local já visitado na vizinhança do próximo passo. Dessa forma, construir réplicas desse tipo de caminhada seria praticamente impossível quanto maior o número de passos n tomados.

Apesar disso, uma abordagem alternativa para o problema pode ser proposta. Podemos iniciar pela construção de uma caminhada autoexcludente e tentar rearranjar sua forma para obtermos as demais. Assim, cada forma rearranjada seria uma aproximação da mesma caminhada.

Com a intenção de obtermos tais resultados apresentaremos neste capítulo dois algoritmos que realizam o rearranjamento das caminhadas: o *Slithering Snake* e o pivô.

[1] Abreviação do termo em inglês *Self-Avoiding Walk*.

3.1 Algoritmo *Slithering Snake*

O algoritmo pode ser dividido em seis partes. Os passos de **2** a **4** são repetidos várias vezes. A primeira vez usando a configuração da SAW inicial dada pela passo **1**, que cria uma caminhada autoexcludente. Então, repetimos os passos de **2** a **4** m vezes, usando a configuração da SAW obtida na execução anterior. Por fim, calculamos o valor médio do quadrado da distância de ponta a ponta.

3.1.1 Descrição do algoritmo *Slithering Snake*

A seguir, vamos descrever passo a passo o algoritmo:

- **Primeiro passo**: criamos uma caminhada autoexcludente de n passos em uma rede quadrada bidimensional, em que todos os passos estão alinhados. Esse primeiro passo do algoritmo cria a configuração inicial de uma caminhada. Por exemplo, para $n = 6$ passos, com o ponto de partida na origem do plano xy, a SAW inicial seria:

  ```
  {{0, 0}, {1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}}.
  ```

  Após isso, calculamos o quadrado da distância de ponta a ponta para SAW inicial;

- **Segundo passo**: selecionamos aleatoriamente um passo, dentre os passos $\{1, 0\}$, $\{-1, 0\}$, $\{0, 1\}$ e $\{0, -1\}$, e, então, adicionamos esse passo ao último elemento da SAW anterior, a fim de produzirmos um novo passo;

- **Terceiro passo**: checamos se o novo passo gerado coincide com algum ponto da configuração anterior. Se sim, invertemos a sequência e o resultado é uma nova caminhada. Caso contrário, excluímos o primeiro passo, a fim de termos o mesmo número de passos ao final do processo, e acrescentamos o novo ponto ao fim da lista para obtermos a nova caminhada. Por exemplo: vamos considerar a SAW inicial como sendo a lista de passos abaixo:

  ```
  {{0, 0}, {1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}}.
  ```

  Se o passo aleatório é {-1, 0}, ao somá-lo ao último elemento da lista, temos que o novo ponto é {5, 0}, que coincide com um elemento da SAW inicial. Dessa forma, a nova SAW é:

  ```
  {{6, 0}, {5, 0}, {4, 0}, {3, 0}, {2, 0}, {1, 0}, {0, 0}}.
  ```

  Por outro lado, se o passo aleatório fosse {0, -1}, o novo ponto seria {6, -1}, que não coincide com nenhum ponto da lista inicial. Então, descartamos o primeiro elemento e acrescentamos o novo ponto ao fim da lista. A SAW obtida fica assim:

  ```
  {{1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0},{6, -1}};
  ```

- **Quarto passo**: calculamos o quadrado da distância de ponta a ponta da SAW obtida;

- **Quinto passo**: executamos a sequência de passos de **2** a **4** m vezes, começando com a SAW inicial;
- **Sexto passo**: calculamos a média dos valores obtidos no passo **4**.

Seguindo as regras do algoritmo *Slithering Snake*, foi possível criar o programa *SlitheringSAW*, que calcula o valor médio do quadrado da distância de ponta a ponta de uma SAW.

3.1.2 Alguns aspectos sobre o algoritmo

Se olharmos como a média do quadrado da distância de ponta a ponta de uma caminhada autoexcludente de n passos muda em função do número m de configurações rearranjadas, notaremos que, para m pequeno, a mudança na dimensão da caminhada é apenas levemente afetada, enquanto que, para m grande, ocorre uma mudança substancial.

Vamos considerar uma caminhada de 10 passos, por exemplo. Obtemos para $m = 1$ e $m = 100$ os seguintes valores médios para o quadrado da distância de ponta a ponta: $91,0$ e $33,7$ respectivamente. Observamos, portanto, que só há mundaças significativas para m grande. Além disso, podemos ressaltar alguns problemas em relação ao algoritmo. É possível uma SAW encontrar-se em uma forma que não pode ser desfeita por si só como mostra a figura 3.1. Isso ocorre porque o método usado é não *ergódico*[2]. No entanto, foi

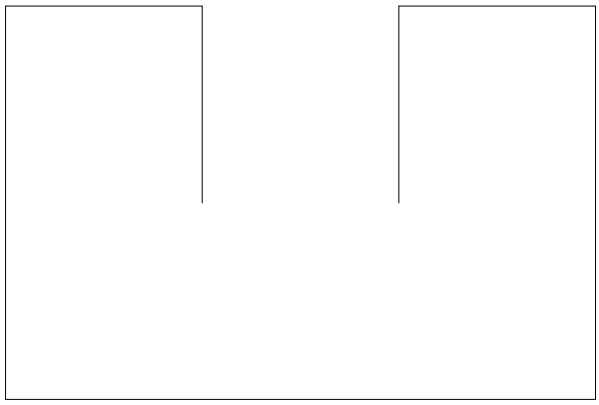

Figura 3.1: Exemplo de uma situação em que a configuração da SAW não permite que a caminhada seja rearranjada pelo algoritmo *Slithering Snake*.

verificado que o expoente crítico obtido pelo algoritmo é correto.

Um outro problema é que, devido ao fato de apenas um ponto ser mudado por vez, o rearranjamento das caminhadas torna-se lento e um grande número de passos deve ser feito na simulação para que se obtenha grandes mudanças na forma da caminhada. Por isso, apresentaremos um outro algoritmo, o algoritmo pivô.

[2]Um processo é dito ser *ergódico* se alguma sequência ou amostra significante é igualmente representativa de um todo.

3.2 Algoritmo Pivô

O algoritmo pivô é muito eficiente para gerar SAWs em d-dimenões em um *ensemble canônico*[3], isto é, com o número de passos fixos. Além disso, ele não possui as limitações do algoritmo *Slithering Snake*. Ele está baseado na seleção aleatória de operações simétricas $2^d d!$ (rotação e reflexão) em uma rede d-dimensional e na aplicação dessas operações na rede. Em duas dimensões, para assegurarmos a ergodicidade, é suficiente considerarmos apenas 3 das 8 operações simétricas. As operações são rotações de $+90$, -90, e 180 graus, que podem ser definidas por matrizes, como mostrado a seguir:

$$R_{90} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad R_{180} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad R_{270} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}. \tag{3.1}$$

A aplicação das operações de rotação usa três medidas: A localização do ponto pivô (torção) ao longo da cadeia; as coordenadas do local do passo n-ésimo da cadeia em relação ao ponto de articulação; e as coordenadas finais do passo n-ésimo da cadeia após a rotação.

3.2.1 Descrição do algoritmo pivô

A seguir, vamos descrever o algoritmo pivô passo a passo:

- **Primeiro passo**: criamos uma SAW em uma rede quadrada bidimensional de n-passos. Ou seja, criamos uma tabela de pares ordenados. Por exemplo, para $n = 6$ temos:

  ```
  {{0, 0}, {1, 0}, {2, 0}, {3, 0}, {4, 0}, {5, 0}, {6, 0}}.
  ```

  Então, calculamos o valor do quadrado da distância de ponta a ponta da SAW criada;

- **Segundo passo**: escolhemos um ponto k aleatoriamente para ser o pivô (com $0 < k < n$) ao longo da SAW e dividimos a caminhada em duas partes. Uma das partes consiste em passos incluindo k que chamaremos de parte fixa. A outra parte consiste nos passos subsequêntes a k, que chamaremos de parte móvel;

- **Terceiro passo**: escolhemos aleatoriamente uma operação de rotação;

- **Quarto passo**: aplicamos a operação escolhida em **3** na parte móvel da etapa **2**, que será chamada de parte nova;

- **Quinto passo**: checamos se algum passo na parte nova coincide com os pontos da parte fixa. Se não coincide, juntamos as duas partes, que nos dão a nova configuração dos passos da SAW e, então, calculamos o quadrado da distância de ponta a ponta. Se coincide, apenas calculamos o quadrado da distância da configuração da SAW anterior;

- **Sexto passo**: repetimos a sequência de **2** a **5** m vezes, começando com a configuração inicial;

[3]Para saber mais sobre *ensemble canônico* veja as referências [9, 10, 11].

- **Sétimo passo**: determinamos a média dos valores obtidos no quinto passo, ou seja, a média do quadrado da distância de ponta a ponta.

3.2.2 Alguns resultados

Notamos, como já esperávamos, que, para pequenos valores de m, é produzido uma grande mudança na forma da caminhada, o que difere esse algoritmo do anterior, no qual necessitávamos de m grande. Em nossas simulações, temos que o valor médio do quadrado da distância de ponta a ponta de uma SAW de 10 passos, calculado pelo programa *Pivot2DSAW* é, para $m = 1$, $< r^2 >= 76,0$, que, no caso do algoritmo *Slithering Snake* , o valor calculado era igual a $91,0$. Por outro lado, com m grande, os valores obtidos por ambos algoritmos foram aproximadamente iguais.

3.3 Dimensão fractal e expoente crítico da caminhada autoexcludente

Uma propriedade de uma SAW, que pode ser considerada muito interessante, é a chamada *dimensão fractal* da caminhada, d_f, dada pela relação $< r^2 >^{1/2} \propto n^{1/d_f}$. As caminhadas autoexcludentes também possuem um expoente crítico ν, dado pela relação $< r^2 > \propto n^{\nu}$, e segue que $d_f = 2/\nu$ para SAWs ou caminhadas aleatórias. A dependência dimensional de uma caminhada aleatória e uma SAW são diferentes. A dimensão fractal de uma caminhada aleatória não depende da dimensionalidade da caminhada, i.e., uma caminhada aleatória executada em d-dimensões tem $d_f = 2$. Entretanto, para uma SAW, a d_f é dependente da dimensionalidade da caminhada.

Segundo Madras e Slade [8], para uma SAW de d-dimensões com $(d \leq 4)$, temos que $< r^2 > \propto n^{6/(d+2)}$. Assim, $\nu = 6/(d+2)$ e $d_f = (d+2)/3$. Para nossas simulações, com n de 10 a 100 com incremento de 5 em 5 passos, sendo feitas $m = 1000$ réplicas para cada caso, encontramos um valor para $\nu \approx 1,47$ bem próximo ao valor esperado para uma caminhada em duas dimensões, $\nu = 1,5$. Além disso, encontramos o valor de $d_f \approx 1,36$ próximo ao valor encontrado na literatura, $d_f = 4/3$. Na figura 3.2, temos o gráfico do valor médio do quadrado da distância de ponta a ponta versus o número n de passos, feito a partir dos dados obtidos em nossa simulação.

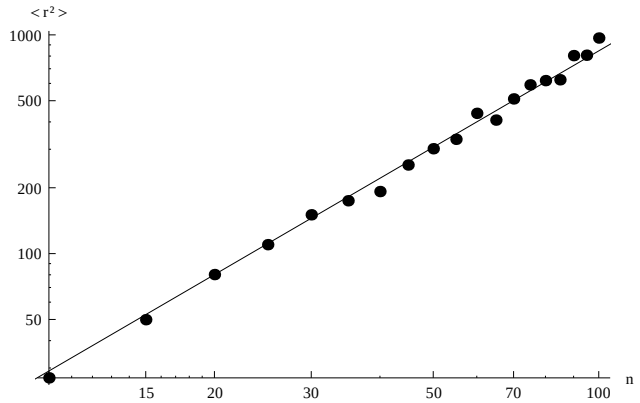

Figura 3.2: Reta obtida por um ajuste linear do logaritmo da lista dos pares ordenados gerados, com $\nu \approx 1,463$. Os pontos são os pares ordenados em escala log-log para valores de n de 10 a 200 e incrementos de 5 em 5, com 1000 caminhadas para cada caso.

Capítulo 4

Acreção

O processo de acreção consiste em partículas se movendo em uma região até que ocorra o encontro entre elas. Ao se encontrarem, as partículas se juntam e formam um algomerado e, então, o processo é repetido de forma que ele cresça até um dado tamanho. Esse aglomerado é formado de maneira irregular, devido ao fato de o processo ser aleatório. Se os aglomerados se formam sem estarem presos a alguma superfície, o processo é chamado de *agregação*, enquanto que os aglomerados formados junto a uma superfície são referidos como *deposição*. Tendo em vista isso, apresentaremos modelos que tratam, separadamente, os dois casos.

Primeiramente, trataremos a agregação por difusão limitada (DLA), na qual uma partícula executa um movimento aleatório até que encontre outra e se juntem. O processo é repetido por várias partículas, levando à formação de um aglomerado. Esse processo é encontrado em muitos sistemas naturais, como na cristalização, na condensação polimérica e coloidal, na formação de fuligem e na formação de nuvens [1, 13, 14, 15].

Por outro lado, há uma variedade de fenômenos que são modelados pela deposição balística. Exemplos que utilizam esse modelo podem ser encontrados na área de materiais, tais como na formação de filmes finos e na deposição de vapor [1, 13, 16]. O processo de deposição consiste nas seguintes etapas: uma partícula inicia um movimento de queda, em cima de um substrato sólido ou alguma superfície, movimento que persiste até que ela encontre a superfície ou outra partícula na qual irá ficar presa.

A seguir, apresentaremos os modelos DLA e de deposição balística. Discutiremos, em cada caso, os algoritmos usados em nossas simulações.

4.1 Modelo de agregação por difusão limitada

Em termos físicos, o modelo DLA pode ser facilmente descrito. Durante qualquer tempo no crescimento de uma DLA, temos uma partícula executando uma caminhada aleatória nas vizinhanças de um aglomerado, onde poderá se agregar. O crescimendo de uma agregação por difusão limitada percorre os seguintes passos: (i) no instante inicial do processo, o aglomerado possui apenas uma partícula; (ii) em uma localização escolhida aleatoriamente, uma partícula começa a executar um movimento aleatório na circunvizinhança do aglomerado inicial, então, se ela atinge uma distância maior que um certo raio dado, a partícula é deseprezada, e se ela é adjacente ao aglomerado, temos a

junção dela com o aglomerado. Esses passos são repetidos por cada partícula até que o aglomerado alcance um tamanho desejado. A seguir, iremos nos aprofundar em alguns detalhes na construção do algoritmo DLA.

4.1.1 Algoritmo DLA

A agregação por difusão limitada pode ser estudada em uma rede quadrada bidimensional, por isso construiremos o algoritmo para uma DLA em duas dimensões, dividindo o processo em cinco passos:

- **Primeiro passo**: criamos uma lista dos sítios ocupados, contendo o ponto {0, 0}. Dessa forma, a lista inicial se apresenta assim:

  ```
  {{0, 0}},
  ```

  que representa a localização da partícula que forma o aglomerado inicial.

- **Segundo passo**: delimitamos a região onde a partícula fará a caminhada. Essa região consiste em um círculo de raio R, cujo valor é dado pela relação $R = s + l$, sendo s um parâmetro dado e l o valor absoluto da localização da partícula mais externa que pertence ao aglomerado, isto é, a partícula agregada que se encontra na região mais distante da partícula inicial em {0, 0}. Feito isso, escolhemos aleatoriamente um ponto dentro do círculo para iniciarmos a caminhada da partícula;

- **Terceiro passo**: começamos uma caminhada aleatória a partir do ponto selecionado no passo anterior do algoritmo. Essa tarefa consiste em sortear um passo aleatório ($\{1, 0\}, \{0, 1\}, \{-1, 0\}$ ou $\{0, -1\}$) e somá-lo ao anterior, repetindo o processo n vezes. Esse processo, que gera caminhadas aleatórias, foi melhor discutido na seção 2.2. Entretanto, aqui, em vez de tomarmos o ponto $\{0, 0\}$ como sendo o início da caminhada, selecionamos a localização escolhida no passo **2**;

- **Quarto passo**: devemos criar um mecanismo que pare a caminhada, visto que não é dado um valor para n. Esse mecanismo consiste em verificar se o passo é adjacente ao aglomerado ou se o passo dado é maior que o quadrado do raio da região circular que delimita o início da caminhada (área delimitada no passo **2**). No primeiro caso, adicionamos o passo à lista de sítios ocupados, no segundo, descartamos a partícula;

- **Quinto passo**: repetimos as operações de **2** a **4** até que o número de sítios ocupados seja um valor dado m.

Por meio desse algoritmo, construímos o programa *DLA*. Além das etapas descritas, adicionamos um contador para calcular o número de partículas lançadas durante o crescimento do processo. Um resultado típico para uma DLA, gerada pelo programa *DLA*, com $s = 2$ e $m = 8$ sítios ocupados, é:

```
O número de partículas lançadas foi 20.
{{0, 0}, {0, 1}, {1, 1}, {1, 0}, {1, 2}, {1, 3}, {0, 3}, {2, 3}}.
```

Note que foram lançadas 20 partículas para que obtivéssemos 8 sítios ocupados. A lista gerada nos dá a localização das partículas no aglomerado. Entretanto, seria interessante se, em vez de olharmos para a lista, pudéssemos ver o aglomerado em uma forma gráfica. Por isso, iremos apresentar algumas formas de visualizarmos uma DLA.

4.1.2 Visualizando uma DLA

Por meio do programa *DLA* e com o *ShowDLA*, criamos a figura 4.1. Os sítios ocupados pelas partículas são repesentados por quadrados e os números em cada quadrado representam a ordem em que cada partícula foi agregada ao aglomerado. Além disso, a cor muda com a distância em relação à primeira partícula do aglomerado, quanto mais longe, mais claro o quadrado.

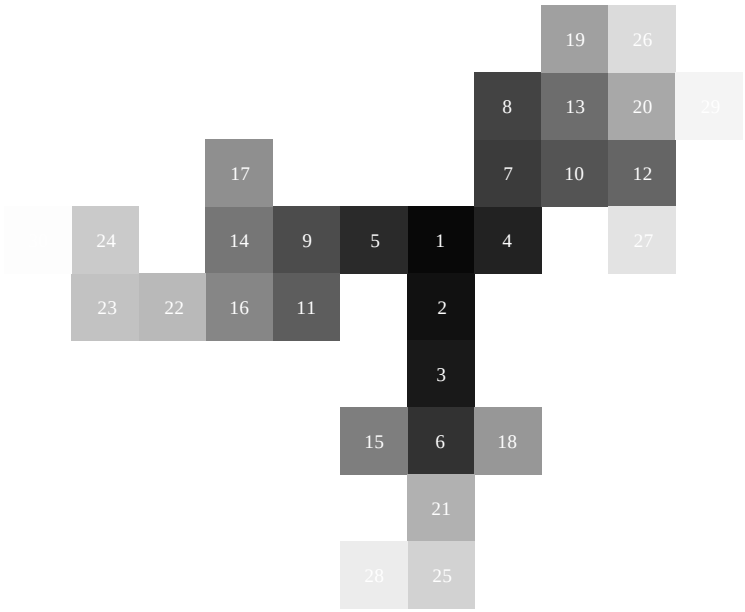

Figura 4.1: Representação gráfica de uma DLA gerada pelo programa *DLA* com $s = 10$ e $m = 30$ sítios ocupados.

Também criamo o programa *ShowDLA2* com o qual construímos a figura 4.2. O procedimento é bem parecido, apenas alguns recursos gráficos foram mudados.

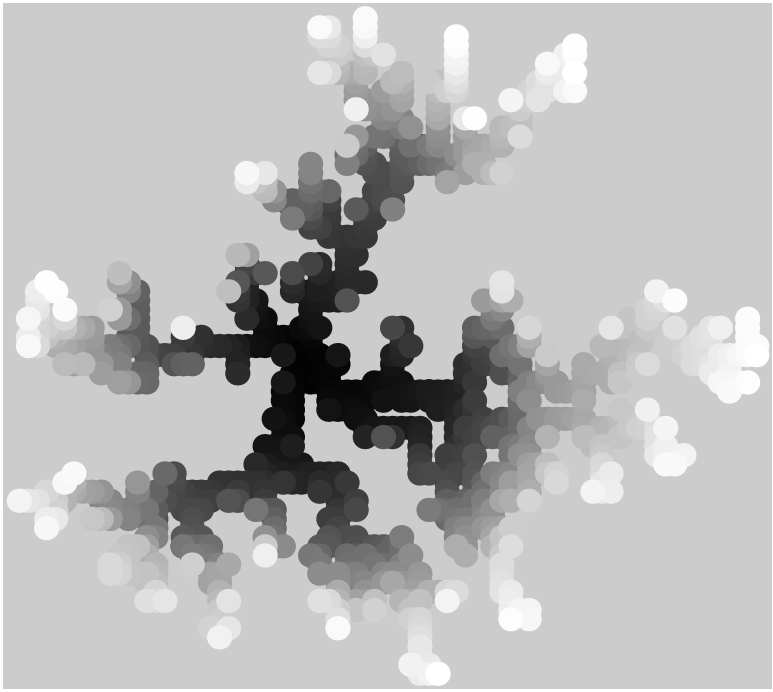

Figura 4.2: Representação gráfica de uma DLA gerada pelo programa *DLA* com $s = 2$ e $m = 1000$ sítios ocupados.

4.2 Dimensão fractal de uma DLA

No processo de formação do aglomerado de uma DLA, o formato da agregação é muito irregular. Isso ocorre porque a probabilidade de uma partícula se agregar a uma porção externa do aglomerado é maior que a porções internas. É difícil uma partícula adentrar se entre as ramificações do aglomerado, a fim de atingir porções mais internas, sem que entre em contato com as partículas que estão agregadas às ramificações do aglomerado. Esse efeito pode ser notado olhando as localizações dos sítios do aglomerado como uma função do momento em que a partícula se junta ao aglomerado.

Para termos uma ideia de como uma DLA preenche o espaço, podemos medir sua dimensão fractal, que pode ser feita de várias formas, por exemplo: a dimensão fractal como dependência em relação ao raio de giração do aglomerado. Entretanto, para todos os tipos de medidas, parece existir uma universalidade, a saber, de acordo com o crescimento do tamanho da DLA, a dimensão fractal diminui. Para nossa análise, vamos olhar para a densidade de espaços ocupados pelas partículas do aglomerado em função do tamanho deste.

O processo de uma DLA é estocástico[1] e, portanto, é necessário tomarmos a média de uma certa quantidade de valores da dimensão fractal de várias DLAs. No entanto, vamos considerar uma única configuração de DLA apenas para fins ilustrativos.

4.2.1 Calculando a dimensão fractal

Para calcularmos a dimensão fractal, devemos seguir os seguintes passos:

[1]Variáveis aleatórias cujos valores variam com o tempo [9].

- **Primeiro passo**: criamos uma tabela de pares ordenados da forma {tamanho do quadrado, densidade de sites ocupados};

- **Segundo passo**: calculamos o coeficiente da inclinação da reta gerada pelos pares ordenados da lista em escala log-log, que é a dimensão fractal.

Com isso, escrevemos o programa *FractalDimension* e calculamos a dimensão fractal de uma DLA feita pelo programa *DLA* com $s = 10$ e $m = 30$ sítios. Segue abaixo um resultado típico obtido pelo programa para esses parâmetros:

```
O número de partículas lançadas foi 138.
A dimensão fractal da DLA é -0.991708.
```

A inclinação negativa obtida para a dimensão fractal significa que a densidade da estrutura da DLA é maior no centro e menor nas periferias.

4.3 Modelo de deposição balística

O modelo de deposição balística pode ser facilmente descrito em termos físicos. Primeiro, consideramos uma superfície, inicialmente lisa, que possuia algumas fileiras de partículas. Em seguida, uma partícula é lançada, a partir de uma posição escolhida aleatoriamente, a uma certa distância acima da superfície. Então, a partícula executa um movimento de queda em linha reta rumo à superfície até que a encontre ou até ser adjacente a outra partícula. Quando ocorre um desses casos, a partícula permance nessa localização e outra partícula é lançada. O processo é repetido para um número dado de partículas desejado. A seguir, temos o algoritmo que executa esse processo.

4.3.1 Algoritmo de deposição balística

Dividiremos o algoritmo de deposição balística em sete passos de execução. O modelo é executado em uma rede quadrada bidimensional e os passos são repetidos por uma quantidade de vezes para obtermos uma deposição com um certo número dado de partículas, como veremos abaixo:

- **Primeiro passo**: inicialmente criamos uma matriz $2 \times n$, na qual a primeira linha é preenchida inteiramente por zeros e a última por vários números uns. Por exemplo, para $n = 5$, teríamos uma matriz da forma:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}; \tag{4.1}$$

- **Segundo passo**: escolhemos aleatoriamente um sítio da primeira fileira da matriz. Esse valor indica em que coluna a partícula irá executar o movimento de queda;

- **Terceiro passo**: usando o valor escolhido no passo anterior, que indica a coluna onde a partícula fará o movimento de queda, criamos uma lista incluindo a coluna selecionada da matriz e as suas colunas vizinhas mais próximas, à direita e à esquerda. Quando a coluna selecionada se encontra em uma extremidada, direita ou esquerda, usamos uma *condição periódica de contorno*. Se a coluna escolhida fosse

a última da direita, tomaríamos a última coluna da esquerda sendo a sua coluna vizinha. Se a coluna escolhida fosse a última da esquerda, a coluna vizinha à esquerda seria a última coluna da direita. Observe que na nova lista feita nessa etapa, temos apenas 3 colunas, a coluna onde a partícula se movimenta e as colunas vizinhas;

- **Quarto passo**: encontramos a posição do primeiro sítio ocupado em cada uma das três colunas listadas e determinamos a que ocorre primeiro. Então, calculamos a posição adjacente ao site ocupado que ocorre primeiramente (posição que está localizada na coluna onde a partícula executou o movimento de queda), conforme determinado anteriormente. Esta será a posição onde a partícula ficará depositada;

- **Quinto passo**: a posição calculada no passo anterior, na representação matricial, está preenchida pelo valor 0. O que devemos fazer é trocar esse valor por 1 para indicarmos que nessa posição temos uma partícula;

- **Sexto passo**: verificamos se a primeira linha da matriz está totalmente preenchida por zeros. Caso isso não seja satisfeito, adicionamos uma linha preenchida por zeros no topo da matriz. Por exemplo, vamos supor a seguinte matriz:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{4.2}$$

  Como podemos notar, a primeira linha não está totalmente preenchida por zeros. Em razão disso, devemos adicionar a ela uma nova linha preenchida por zeros. Assim, a nova matriz obtida por esse procedimento é:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}; \tag{4.3}$$

- **Sétimo passo**: executamos os passos de **2** a **6** repetidas vezes até obtermos t sítios ocupados.

Por meio desse algoritmo, construímos o programa *MolecularDeposition*, no qual os valores de entrada são o número n de partículas iniciais depositadas e o valor t de sítios ocupados desejados. Para $n = 5$ e $t = 10$, um resultado típico obtido pelo programa, na forma matricial, é:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{4.4}$$

4.3.2 Visualizando a deposição balística

Com o resultado obtido pelo programa, podemos construir uma representação gráfica para a deposição balística de partículas. Assim, por meio do programa *ShowDeposition*, geramos a figura 4.3 via os dados obtidos em nossas simulações, tomando $n = 50$ partículas iniciais e o número t de partículas depositadas igual a 1000.

Figura 4.3: Representação gráfica de uma deposição com $n = 50$ partículas iniciais e $t = 1000$ partículas depositadas.

Capítulo 5

Fenômenos de Espalhamento

Quando nos referimos a fenômenos de espalhamentos, alguns exemplos do nosso cotidiano podem ser recordados, tais como quando derrubamos um copo com líquido ou o cheiro do perfume de alguém que acabou de chegar. De fato, existem vários fenômenos na natureza que possuem esse comportamento de espalhar-se em uma dada região do espaço, possuindo um local com maior concentração e estendendo-se por uma grande área em diferentes concentrações. Exemplos de sistemas naturais descritos pelo modelo de espalhamento ou crescimento cinético (KG[1]) incluem crescimento de tumores [17], congelamento [18], disseminação de epidemias [19], espalhamento de rumores [20] e fluxo de um fluido em um meio poroso [21, 22, 23].

O modelo KG foi originalmente introduzido por um biológo e denominado *Modelo de Eden* [17] para representar o crescimento de tumores. O modelo descreve um sistema em uma rede quadrada bidimensional com um único aglomerado inicialmente, uma *semente*, localizado na origem. Então, escolhemos um sítio aleatoriamente (dentre os que se encontram no perímetro da localização da semente) para que seja parte do aglomerado. Em seguida, temos a exclusão desse ponto da lista de pontos que fazem parte do perímetro. Depois, ele é incluído dentro da lista de localização do aglomerado. Por fim, os pontos mais próximos, que cercam o ponto escolhido, são incluídos na lista de pontos perimetrais ao aglomerado. O processo é repetido até que o aglomerado tenha um número n de sítios.

Veremos duas variações do modelo de Eden: percolação aleatória e percolação por invasão.

5.1 Modelo de percolação aleatória

O modelo de percolação aleatória de aglomerados descreve processos tal como a disseminação de uma doença. No processo descrito pelo modelo, temos, inicialmente, em uma rede quadrada bidimensional, uma semente no ponto $\{0, 0\}$, que dará início ao espalhamento. Criamos uma lista com os pontos que fazem parte do aglomerado e outra com os pontos que pertencem ao perímetro do aglomerado. Cada sítio perimetral ao algomerado possui uma probabilidade p de juntar-se a ele. Então, para que ocorra o crescimento do aglomerado, escolhemos um sítio aleatoriamente e, de acordo com uma regra de seleção, novos sítios são agregados ao aglomerado. Esse procedimento ficará claro durante a cons-

[1]KG, do termo em inglês *Kinetic Growth*.

trução das tarefas que compõem o algoritmo de uma percolação aleatória. Se tomarmos $p = 1$, o modelo se reduz ao modelo de Eden.

5.1.1 Algoritmo de percolação aleatória

Iremos descrever o algoritmo que simula a disseminação de uma doença, de acordo com o modelo de percolação aleatória. Dividiremos o algoritmo em quatro etapas, sendo que a terceira delas possui algumas subetapas. Estas subetapas nos dão uma regra de seleção para adicionarmos novos sítios ao aglomerado.

- **Primeiro passo**: criamos um par ordenado contendo a lista dos sítios do aglomerado e a lista perimetral. Enquanto aquela possui apenas uma semente que está localizada no ponto {0, 0}, esta possui quatro sítios para essa configuração inicial. Portanto, temos um par ordenado da forma {lista do aglomerado, lista perimetral} que, para o instante inicial, é:

  ```
  {{{0, 0}},{{1, 0}, {0, 1}, {-1, 0}, {0, -1}}}.
  ```

  Também criamos uma lista vazia chamada rejeitados;

- **Segundo passo**: escolhemos aleatoriamente um sítio na lista perimetral;

- **Terceiro passo**: geramos um número aleatório η (com $0 \leq \eta \leq 1$) e comparamos com p. Se o valor do número aleatório for menor ou igual a p, seguimos o passo **A**, caso contrário, seguimos o **B**:

  Passo A: determinamos os sítios vizinhos mais próximos ao sítio selecionado. Então, determinamos quais deles não se encontram na lista perimetral, ou na do aglomerado ou na de rejeitados e adicionamos tais sítios à lista perimetral. Feito isso, incluímos o sítio selecionado na lista do aglomerado e adicionamos à lista perimetral os sítios vizinhos a ele que ainda não estão em nenhuma lista. Por fim, atualizamos o par {lista do aglomerado, lista perimetral}, que agora possui novos sítios;

  Passo B: colocamos o sítio selecionado na lista de rejeitados. Retiramos o sítio selecionado da lista de sítios perimetrais e, então, atualizamos o par {lista do aglomerado, lista perimetral}, que agora possui a nova lista perimetral e a antiga lista do aglomerado;

- **Quarto passo**: executamos a sequência de passos **2** e **3** até o tamanho do aglomerado atingir um valor n ou a lista perimetral estiver vazia.

Por meio desse algoritmo, construímos o programa *Epidemic*. Na seção 5.3, mostraremos algumas representações gráficas obtidas em nossas simulações.

Além desse programa, também construímos o *Eden* que é relativamente mais rápido que o *Epidemic*. Porém, o processo só é feito por ele com $p = 1$, não sendo possível trocar esse valor. Contudo, se estivermos interessados apenas nesse caso, é preferível usá-lo, eliminando as tarefas desnecessárias do algoritmo. No caso da criação de aglomerados de *Eden*, não é necessário ter uma lista de sítios rejeitados, nem a condição usada para determinar se o valor aleatório é maior que a probabilidade p e, assim, escolher a operação que será executada. É por esse motivo que o programa *Eden* é mais rápido no caso em que $p = 1$.

5.2 Percolação por invasão

O modelo de percolação por invasão descreve o fluxo de um fluido através de um meio poroso [21, 22]. Nesse modelo, cada sítio na lista perimetral tem um número aleatório associado. Além disso, o crescimento é dado pelo espalhamento do aglomerado que incorpora os sítios perimetrais com menor número aleatório associado. Esse modelo descreve um processo de espalhamento que segue caminhos onde há a menor resistência para o fluido escoar [1]. O modelo cresce em uma rede quadrada bidimensional e o algoritmo que gera a percolação por invasão é descrito a seguir.

5.2.1 Algoritmo de uma percolação por invasão

Este algoritmo pode ser dividido em seis passos, a saber:

- **Primeiro passo**: criamos um par ordenado no qual o primeiro componente é a lista de sítios do aglomerado contendo a semente, o ponto {0, 0}. O segundo componente é a lista perimetral que é constituída de pares ordenados da forma {número escolhido aleatoriamente, vizinho mais próximo do aglomerado};

- **Segundo passo**: selecionamos o sítio, dentro da lista perimetral, com o menor número aleatório associado;

- **Terceiro passo**: determinamos quais são os sítios na vizinhança do ponto selecionado que não estão na lista dos sítios que fazem parte do aglomerado ou na lista perimetral. Feito isso, associamos um número aleatório a cada um deles e os listamos na forma de pares;

- **Quarto passo**: removemos o sítio selecionado em **2** e seu número associado da lista perimetral e o adicionamos na lista dos sítios do aglomerado. Então, adicionamos à lista perimetral os sítios vizinhos (com seus respectivos números aleatórios associados) que não fazem parte de nenhuma lista;

- **Quinto passo**: atualizamos o par ordenado do passo um. Para tanto, fazemos com que o primeiro componente possua os pontos da lista de sítios do aglomerado mais o sítio selecionado em **2**, e que a lista perimetral possua os sítios antigos, excluindo o sítio selecionado, mais os vizinhos que não pertencem à lista de aglomerados. Por exemplo: vamos supor que temos o aglomerado inicial com apenas um sítio ocupado, sendo ele o {0, 0}. Logo, os sítios vizinhos são:

  ```
  {1, 0}, {0, 1}, {-1, 0}, {0, -1}.
  ```

  Vamos supor que, após associarmos números aleatórios a cada sítio, o ponto {0, 1} possuia o menor deles. Então, a lista de sítios ocupados nova será:

  ```
  {{0, 0},{0, 1}},
  ```

  e a lista dos sítios vizinhos ao aglomerado será:

  ```
  {{1, 0}, {-1, 0}, {0, -1}, {0, 2}, {-1, 1}, {1, 1}}.
  ```

Observação: vale lembrar que esse exemplo nos mostra apenas a lista da localização dos sítios. Por simplicidade, os números aleatórios associados foram omitidos. Entretanto, para que o processo seja feito por completo, devemos seguir as etapas descritas no algoritmo;

- **Sexto passo**: repetimos os passos de **2** a **5** até que a lista dos sítios do aglomerado alcance um valor n de sítios ocupados.

Com esse algoritmo, criamos o programa *Invasion*, que foi utilizado em nossas simulações, como é mostrado na seção seguinte.

5.3 Representação gráfica de fenômenos de espalhamento

Nessa seção, apresentaremos alguns resultados obtidos em nossas simulações por meio de figuras que representam os fenômenos de espalhamentos. As figuras foram geradas pelos programas *ShowSpread* e *ShowDLA2*.

No primeiro exemplo (figura 5.1), temos a representação gráfica de um resultado típico obtido pelo programa *Eden* ou pelo *Epidemic* usando o parâmetro $p = 1$. No segundo (figura 5.2), cujos dados foram gerados pelo programa *Epidemic*, observamos uma epidemia disseminada em 5000 indivíduos. No terceiro (figura 5.3), usamos o programa *ShowDLA2* para criar uma representação de uma epidemia conforme a história. A ordem temporal é representada de acordo com a cor. No quarto exemplo (figura 5.4), usamos o programa *Invasion* para gerar os dados.

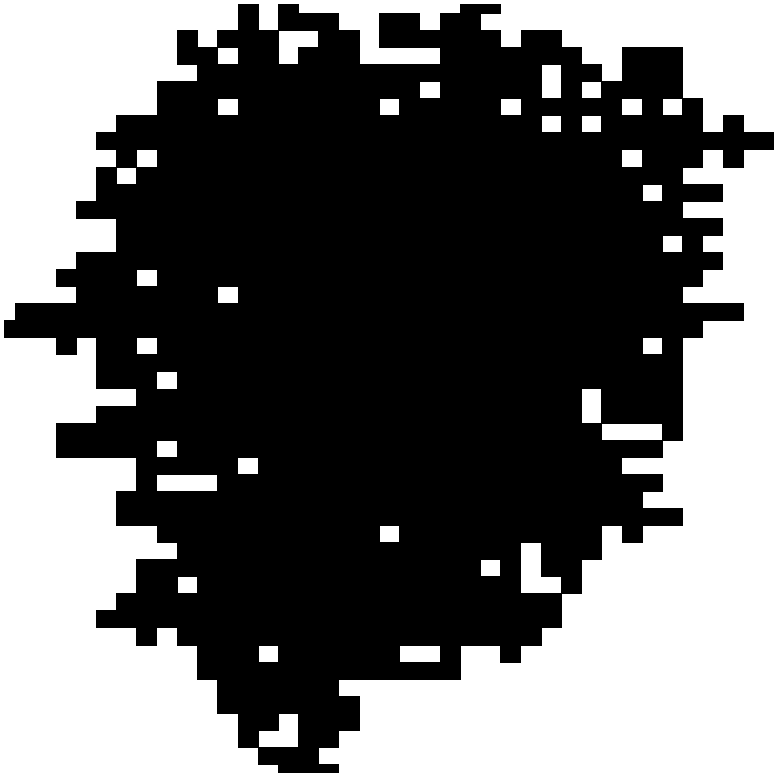

Figura 5.1: Representação gráfica de um espalhamento feito pelo modelo de *Eden* com $n = 1000$ sítios ocupados. O mesmo resultado pode ser obtido pelo modelo de percolação aleatória para $p = 1$.

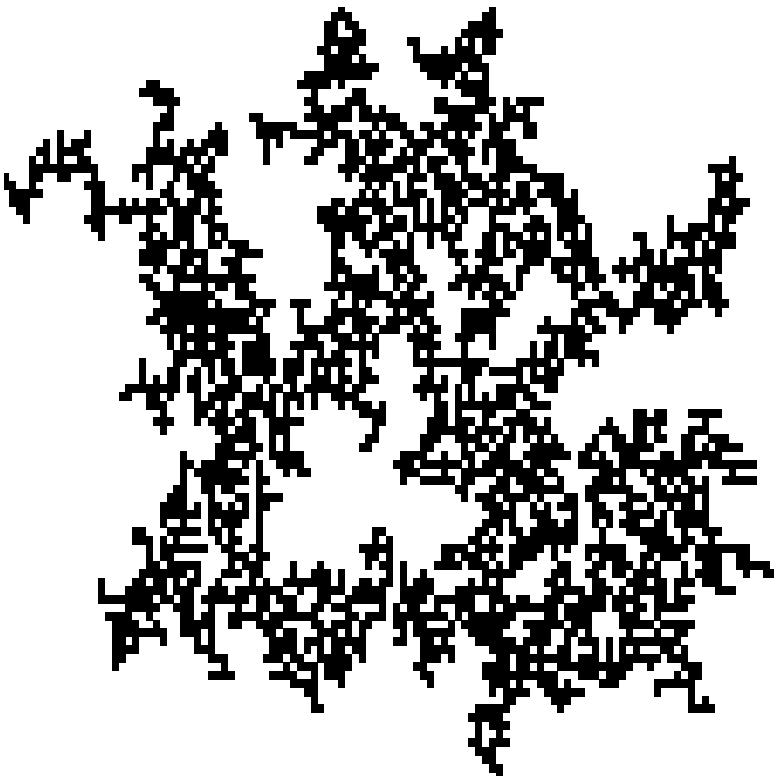

Figura 5.2: Representação gráfica de uma epidemia disseminada em 5000 indivíduos com probabilidade $p = 0,58$ de que sítios perimetrais sejam infectados.

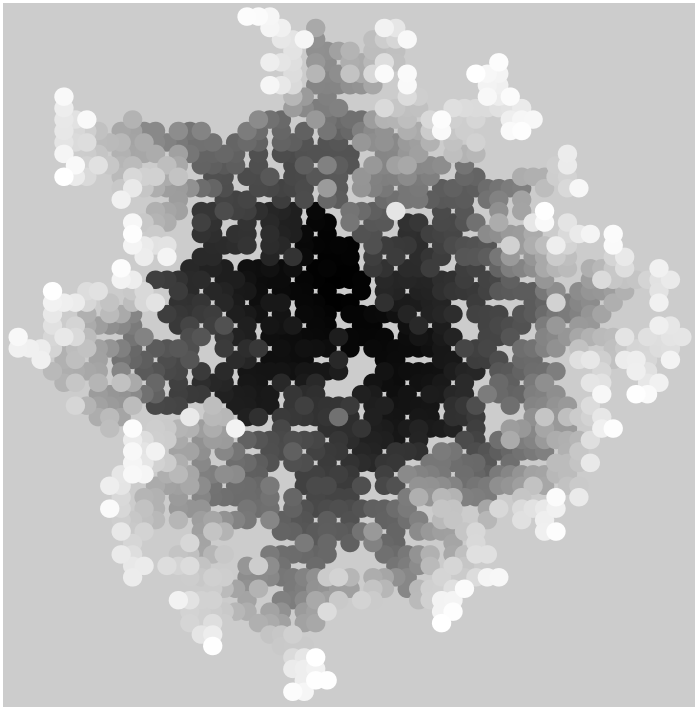

Figura 5.3: Representação gráfica de uma epidemia disseminada em 2000 indivíduos com probabilidade $p = 0,7$ de que sítios perimetrais sejam infectados. As cores representam a ordem temporal de contaminação dos indivíduos, começando no mais escuro rumo ao mais claro.

Figura 5.4: Representação gráfica do fluxo de um fluido em um meio poroso após 5000 sítios serem ocupados.

Capítulo 6

Percolação de Aglomerados

O processo de *transição sol-gel* [24] é o fenômeno de conversão de um líquido em um semissólido. Exemplos desses processos são o coalhar do leite e o coagular do sangue. Assim, se olharmos para o que há em comum em ambos, veremos que eles consistem em moléculas que vão se agrupando aleatoriamente. Essa semelhança é uma característica de um fenômeno de percolação.

Em 1991, o francês Pierre-Gilles de Gennes ganhou o prêmio Nobel em física por seus trabalhos em física teórica sobre materiais desordenados. Em um desses trabalhos, ele descreveu a transição de percolação da seguinte forma [25]: "muitos fenômenos são feitos de ilhas aleatórias e em certas condições, entre essas ilhas, um continente macroscópico emerge".

Na natureza, existe uma grande quantidade de fenômenos de percolação [23, 26], por isso que os processos que envolvem a percolação estão presentes em diversas áreas do conhecimento [1, 23, 26]: em biologia (na reação imunológica anticorpo-antigênio), em química (em uma reação de polimerização) e em física (fenômenos críticos). O comportamento emergente devido à interação entre as partes do sistema é, de fato, crucial para o entendimento desse tipo de processo.

Tendo em vista isso, a modelagem de fenômenos de percolação torna-se interessante do ponto de vista de uma abordagem computacional. Sendo assim, discutiremos, a seguir, o modelo de percolação de sítios aleatórios.

6.1 Percolação de sítios aleatórios

O modelo de percolação de sítios aleatórios consiste em uma rede Booleana aleatória ($m \times m$) em que os sítios possuem valores 0 ou 1: 1 representa um sítio ocupado e o valor 0 um vazio. A probabilidade p de um local estar ocupado não depende das probabilidades de seus vizinhos. O aglomerado é um grupo dos sítios vizinhos adjacentes ocupados.

6.1.1 Algoritmo para simulação de percolação de sítios aleatórios

Um algoritmo simples que simula esse tipo de processo pode ser dividido em duas etapas, como segue:

- **Primeiro passo**: criamos uma matriz $m \times m$ e, para cada elemento da matriz, sorteamos um número aletório η;
- **Segundo passo**: se $\eta \leqslant p$ (em que p é um dado valor), então, o elemento da matriz é preenchido com o número 1 (sítio ocupado). Caso contrário, o elemento é preenchido com 0 (sítio vazio).

Com base nesse algoritmo, construímos o programa *SitePercolation*, que tem como resultado típico, na forma matricial, para os parametros $p = 0,35$ e $m = 10$, a seguinte matriz:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{6.1}$$

Olhando para essa matriz, fica díficil saber onde estão os aglomerados. Entretanto, podemos facilmente identificá-los se fizermos uma representação gráfica dessa matriz, em que os sítios ocupados são quadrados em preto e os vazios são quadrados em cinza. Assim, construímos, por meio do programa *DisplayPercolation*, a figura 6.1 a partir dos dados da matriz 6.1.

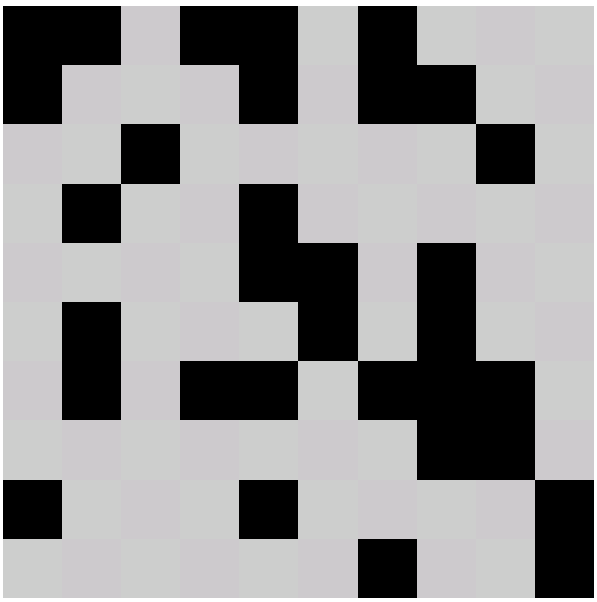

Figura 6.1: Representação gráfica da simulação de uma percolação de sítios aleatórios para os parâmetros $p = 0,35$ e $m = 10$.

Uma outra maneira de identificar os aglomerados seria nomeá-los por números: cada sítio ocupado possuiria o número do aglomerado ao qual pertence. Dessa forma, haveria

um número de quantidades de aglomerados na percolação, que nos levaria a investigação de caracteristicas espaciais, tais como dimensão fractal e percolação *threshold* (limiar). A percolação *threshold* é alcançada quando, para um dado p, forma-se um primeiro aglomerado que se espalha, sem interrupções, pela rede toda. Para essas investigações é necessário um programa que nomeie os aglomerados. Para isso, iremos discutir, na próxima seção, um método bem conhecido: 0 *algoritmo Hoshen-Kopelman* [27].

6.2 Algoritmo Hoshen-Kopelman: nomeação dos aglomerados

Esse algoritmo tem a tarefa de nomear uma dada lista r (a qual pode ser obtida a partir do programa *SitePercolation*) que contém m sublistas, em que cada uma possui m elementos, que podem ser 0 ou 1. A seguir, iremos descrever passo a passo o algoritmo. Além disso, iremos apresentar a descrição de um exemplo específico que nos ajudará a entender o algoritmo. Nesse exemplo, iremos considerar a seguinte matriz:

$$r = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6.2}$$

- **Primeiro passo**: criamos uma lista u a partir da lista r da seguinte forma: em cada uma das m sublistas, adicionamos no início um elemento zero; criamos uma sublista, preenchida por zeros, com $(m+1)$ elementos e a posicionamos como sendo a primeira sublista na lista r. Feito isso, chamamos essa nova lista de u, que, na representação matricial, é uma matriz $(m+1) \times (m+1)$ com a primeira linha e a primeira coluna preenchidas por zeros. Observe que os elementos restantes são os mesmos da matriz r. Assim, considerando a matriz r do nosso exemplo e fazendo o procedimento acima descrito, temos:

$$u = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \tag{6.3}$$

- **Segundo passo**: criamos uma matriz ul com o mesmo tamanho da matriz u, na qual todos os elementos possuem o valor 0. Em nosso exemplo, temos a seguinte matriz:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; \tag{6.4}$$

- **Terceiro passo**: criamos uma lista ulp vazia:

$$ulp = \{\}; \tag{6.5}$$

- **Quarto passo**: escaneamos linha a linha os elementos u_{ij} da matriz u, começando em $i = 2$ e $j = 2$, passando por todos os elementos até que $i = j = m + 1$. Os elementos em ul e ulp são atualizados de acordo com as regras abaixo:
  - **A** Em cada elemento u_{ij}, verificamos se ele está ocupado ou vazio de acordo com as condições
  - **A.1** e **A.2**;
  - **A.1** Se u_{ij} estiver vazio, passamos para o próximo elemento. Repetimos esse procedimento até que o valor do elemento selecionado seja 1. Em nosso exemplo, o primeiro elemento com o valor 0 é o u_{23};
  - **A.2** Se u_{ij} estiver ocupado, devemos olhar os sítios vizinhos mais próximos da coluna anterior e da linha anterior, como descrito em **A.2.1**, **A.2.2** e **A.2.3**. Em nosso exemplo, o elemento u_{22} está ocupado;
  - **A.2.1** Se os vizinhos da coluna e da linha anterior estiverem vazios, estabelecemos que o elemento da matriz ul, o sítio ul_{ij}, é igual a $max+1$, sendo max o maior valor dos elementos de ul. Então, adicionamos esse novo valor máximo de ul em ulp. Em nosso exemplo, notamos que esse é o caso dos elementos u_{22}, u_{25}, u_{45} e u_{55}. Considerando apenas o elemento u_{22}, que é o primeiro elemento do escaneamento, a matriz ul e a lista ulp são atualizadas e ficam assim:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.6}$$

e

$$ulp = \{1\}; \tag{6.7}$$

Nas etapas seguintes do algoritmo, vamos repetindo o procedimento **A.1** até o elemento u_{25}, em que temos a mesma situação de u_{22}. Portanto, temos:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.8}$$

e

$$ulp = \{1, 2\}; \tag{6.9}$$

A.2.2 Se apenas um dos sítios (ou da coluna ou da linha anterior) estiver ocupado, estabelecemos que o elemento ul_{ij} da matriz ul é igual ao valor do elemento de ul correspondente. Em nosso exemplo, isso ocorre no elemento u_{32} de nossa matriz r. Como o elemento u_{22} está ocupado, o ul_{32} deve ser igual ao valor de ul_{22}. Portanto, nossa matriz e nossa lista ficam desta forma:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.10}$$

e

$$ulp = \{1, 2\}; \tag{6.11}$$

– **A.2.3** Se os dois sítios estiverem ocupados (os elementos da linha e da coluna anterior), estabelecemos que o elemento ul_{ij} da matriz ul é igual ao menor valor dentre os elementos de ulp correspondentes aos elementos da linha e da coluna anterior de ul_{ij}. Por exemplo, se os valores de $u_{(i-1)(j)}$ e $u_{(i)(j-1)}$ são, respectivamente, 2 e 5, o valor que escolhemos na lista ulp é o menor entre o segundo e quinto elemento. Após termos feito isso, também mudamos o valor do maior elemento da lista ulp, entre os dois escolhidos, e estabelecemos que o valor dele é o menor dentre os dois. Isso equivale a dizer que, se aquele elemento não fazia parte daquele aglomerado em um passo anterior, agora ele faz, por ser adjacente ao aglomerado de menor valor. Em nosso exemplo, após fazermos o procedimento **A.2.1** para o elemento u_{34}, temos os seguintes resultados:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.12}$$

e

$$ulp = \{1, 2, 3\}; \tag{6.13}$$

Então, nos deparamos com a situação desse passo (**A.2.3**) no elemento u_{35}. Ao atualizarmos a matriz, temos:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{6.14}$$

e

$$ulp = \{1, 2, 2\}. \tag{6.15}$$

Note que ainda existe um sítio com o número 3, mesmo sendo adjacente ao aglomerado de número 2. A renomeação completa e correta é feita no final, no quinto passo. Nos próximos sítios, temos situações idênticas àquelas já apresentadas e, por isso, irei omiti-las, dando apenas o resultado final após a realização dessas etapas, como segue:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 & 2 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix} \tag{6.16}$$

e

$$ulp = \{1, 2, 2, 4\}. \tag{6.17}$$

- **Quinto passo**: renomeamos os sítios em ul de forma que aqueles que estão conectados entre si possuam o mesmo número. Assim, temos as seguintes transformações para o nosso exemplo,

$$\{1 \to 1, 2 \to 1, 2 \to 2, 4 \to 4\} \tag{6.18}$$

que implica o seguinte ressultado:

$$ul = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{pmatrix}. \tag{6.19}$$

Por meio desse algoritmo, construímos um programa chamado *ClusterLabel* para a nomeação dos aglomerados. Na seção seguinte, vamos usá-lo para obter alguns resultados.

6.3 Visualizando os resultados

Por meio dos programas *SitePercolation* e *ClusterLabel*, podemos criar uma matriz que represente uma percolação e nomearmos os aglomerados com números segundo as regras do algoritmo de Hoshen-Kopelman. Considerando a matriz r gerada pelo programa *SitePercolation*, com os parâmetros $p = 0,493$ e $m = 9$, temos o seguinte resultado típico obtido pelo programa *ClusterLabel*:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 4 & 4 & 4 & 0 & 4 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 & 4 & 0 \\ 0 & 0 & 7 & 7 & 0 & 0 & 0 & 4 & 4 & 4 \\ 0 & 8 & 0 & 7 & 0 & 0 & 4 & 4 & 0 & 0 \\ 0 & 8 & 0 & 7 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 8 & 0 & 10 & 0 & 0 & 11 & 0 & 0 \end{pmatrix}; \tag{6.20}$$

A matriz (6.20) representa um sistema de percolação de aglomerados. Contudo, além desse resultado, uma outra forma interessante de olhar a matriz seria por meio de uma representação gráfica. Por isso, contruímos um programa chamado *ShowPercolation* que obtém uma representação gráfica para a percolação de aglomerados, como podemos ver na figura 6.2.

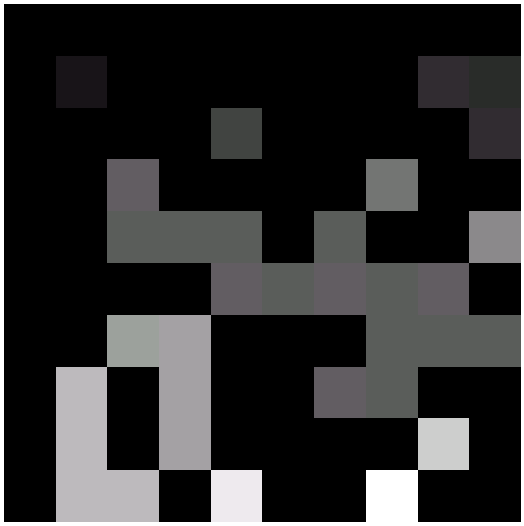

Figura 6.2: Representação gráfica de uma percolação de aglomerados para a matriz (6.20). Note que cada aglomerado está identificado por uma tonalidade diferente da cor cinza.

Outros exemplos são mostrados nas figuras 6.3, 6.4 e 6.5, nas quais temos percolações de aglomerados com diferentes valores de p: respectivamente, $p = 0,2$, $p = 0,3$ e $p = 0,9$. Para essas imagens, representamos os sítios ocupados por quadrados em cor branca e sítios vazios em cor preta. Observe que, com o aumento do valor de p, a tendência é que se forme um único aglomerado.

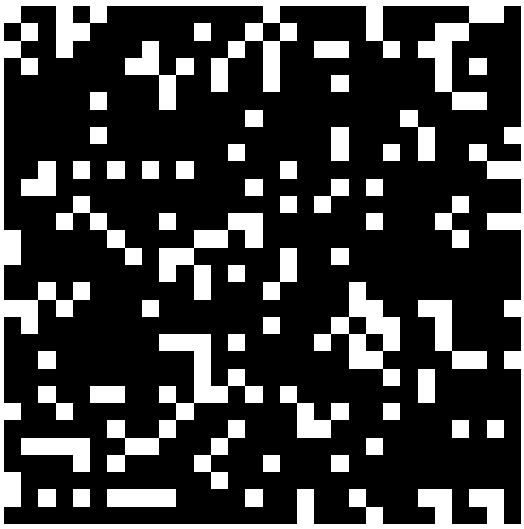

Figura 6.3: Percolação de aglomerados com $p = 0,2$. Os quadros brancos representam os sítios ocupados, enquanto os negros, os vazios.

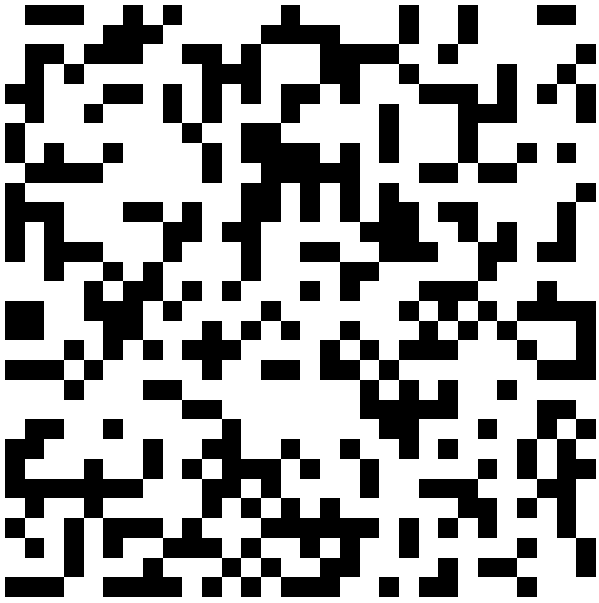

Figura 6.4: Percolação de aglomerados com $p = 0,6$. Os quadros brancos representam os sítios ocupados, enquanto os negros, os vazios.

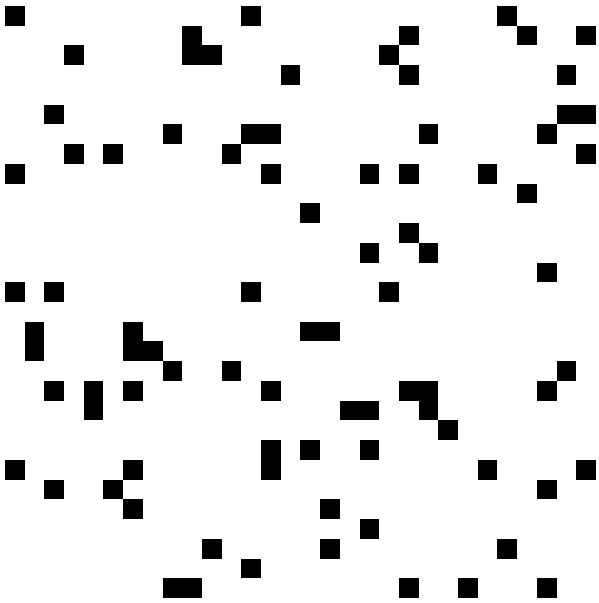

Figura 6.5: Percolação de aglomerados com $p = 0,9$. Os quadros brancos representam os sítios ocupados, enquanto os negros, os vazios.

Capítulo 7

Modelo de Ising

O modelo de Ising pode representar o comportamento de agentes em uma rede. Nesse modelo, cada agente muda seu comportamento de acordo com a ação de seus vizinhos mais próximos. Para investigá-los, empregamos o método metropolis, que é uma versão particular do método de Monte Carlo [28]. Exemplos de contextos nos quais podemos aplicar o modelo de Ising são física do magnetismo, separação de fases em ligações binárias, vidros de spins [9, 10, 11], alinhamento de peixes em um cardume, vaga-lumes piscando em sincronia e até seres humanos seguindo as tendências da moda [29]. Na figura 7.1, temos uma representação do modelo se Ising.

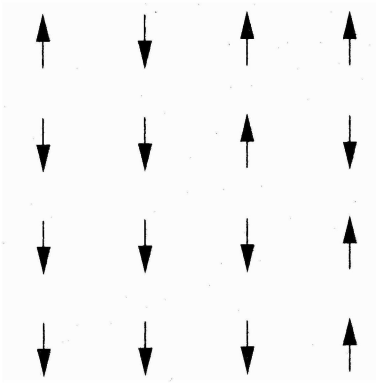

Figura 7.1: Representação pictória do modelo de Ising.

Em nossos estudos, vamos considerar o modelo de Ising probabilístico para a investigação da magnetização de um dado sistema. Para tanto, vamos considerar a formulação do *ensemble canônico* [9, 10, 11], o que implica a temperatura ser constante. A versão do modelo bidimensional consiste em uma rede quadrada $n \times n$, em que cada sítio da rede está associado ao valor 1, para o spin *up*, e -1, para o spin *down*. Os sítios vizinhos, adjacentes a uma dada localização, interagem em pares com uma magnitude J. A menor energia ocorre, para J positivo, quando os spins apontam na mesma direção e, para J negativo, quando estão em posições opostas. Podemos ter um campo magnético externo cujo módulo é igual a B. A diferença entre o número de spins *up* e *down* nos dá

a magnetização do sistema e sua energia é:

$$E = -J \sum_{\{i,k\}} S_i S_k + B \sum_i S_i, \tag{7.1}$$

em que $S_i = \pm 1$ e $\{i, k\}$ representa a soma sobre os primeiros vizinhos.

7.1 Simulação do modelo de Ising

Nesta seção, vamos apresentar um algoritmo que simula o modelo de Ising em uma rede quadrada bidimensional. A sequência de passos é executada um determinado número de vezes, primeiro com a configuração inicial e depois com a obtida na execução anterior.

7.1.1 O algoritmo de Ising

- **Primeiro passo**: criamos uma rede quadrada e sorteamos aleatoriamente, para cada sítio, os valores $+1$ e -1. Portanto, nossa configuração inicial será uma rede com spins orientados de forma aleatória;

- **Segundo passo**: selecionamos aleatoriamente um sítio na rede;

- **Terceiro passo**: determinamos a mudança de energia envolvida na interação entre os spins com o spin do sítio selecionado. Esse processo é dividido em duas partes:

  (1) determinamos quais são os sítios vizinhos ao escolhido. Quando selecionamos sítios do interior da rede, os vizinhos são os quatro elementos que estão mais próximos, isto é, os sítios acima, abaixo, à direita e à esquerda. Por outro lado, se escolhermos um sítio da borda, selecionamos sítios da borda oposta como sendo os vizinhos, isto é, usamos uma condição periódica de contorno;

  (2) a mudança de energia que resulta da interação entre os spins é determinada pela quantidade $2\times$(valor do sítio selecionado)$\times(B + J\times$(spin total dos sítios vizinhos)), sendo B e J valores de entradas dados em unidades de $(1/kT)$, em que k é a constante de Boltzmann e T a temperatura;

- **Quarto passo**: determinamos se a interação inverte ou não o spin selecionado, usando o método metropolis:

  (1) verificamos se existe uma variação negativa de energia como resultado da interação;

  (2) se a variação da energia não é negativa, verificamos se a exponencial do negativo da variação da energia é maior que um número aleatório entre 0 e 1;

  (3) se uma dessas condições é satisfeita, invertemos o spin;

  (4) retornamos à nova configuração, substituindo a inicial por essa nova, com o spin invertido;

- **Quinto passo**: executamos esse processo m vezes;

- **Sexto passo**: criamos uma sublista, a partir da lista com as m configurações, que contém apenas os elementos de n^2 em n^2. Cada elemento dessa lista é correspondente a um passo de *Monte Carlo*, que é o menor valor de inversões que poderíamos fazer

para que tivéssemos escaneado a rede toda. Entretanto, como o processo é aleatório, isso não ocorre necessariamente. Contudo, tomar os elementos de n^2 em n^2 é uma boa aproximação. Para um estudo aprofundado do método de Monte Carlo, veja as referências [9] e [30];

- **Sétimo passo**: calculamos, para cada elemento da lista criada no sexto passo, alguma propriedade global, tal como o valor absoluto da magnetização na rede.

Com esse algoritmo, criamos o programa *IsingMetropolis*, que simula a interação entre os spins.

7.2 Comportamento magnético do modelo de Ising

O comportamento magnético passo a passo pode ser visto em um gráfico por meio do programa *ShowIsingMagnetzation*. Como o decaimento é muito rápido e o sistema chega ao equilíbrio com poucos passos, fizemos algumas modificações para observar melhor o decaimento. Em vez de começarmos com uma rede totalmente aleatória, em que temos a temperatura muito alta, consideramos a rede inicial com todos os spins sendo spins *up*. Também consideramos todas as configurações geradas em vez de apenas as de n^2 em n^2. Observe, na figura 7.2, que se considerássemos apenas as configurações de n^2 em n^2, teríamos apenas dois passos, um em 2500 e outro em 5000, o que dificultaria a vizualização do decaimento. Isso demandaria uma quantidade de passos muito maior e em uma rede maior para chegarmos ao mesmo resultado, além de um enorme esforço computacional.

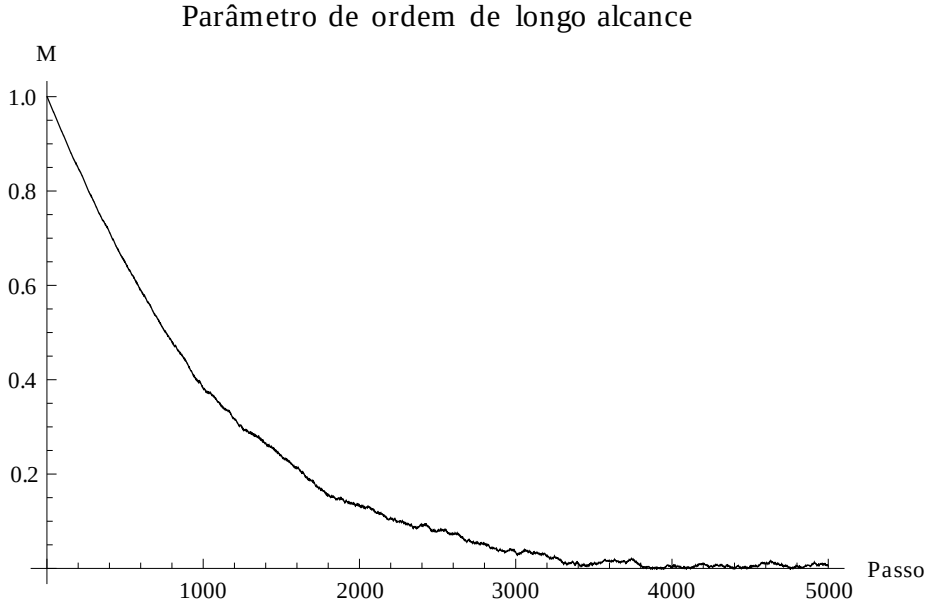

Figura 7.2: Simulação da relaxação da magnetização da rede de spins com os parâmetros $n = 50$ (tamanho da rede), $m = 5000$ (número de inversões feitas), $B = 0$ (campo magnético aplicado) e $J = -0,1$, que é a energia de inversão em unidades de $1/kT$.

Capítulo 8

Evolução Darwiniana

Charles Darwin foi quem mais contribuiu para o nosso conhecimento sobre a evolução biológica das espécies. Atualmente, um ponto fundamental sobre a evolução, que fora apontado primeiramente na teoria Darwiniana [32], é a seleção natural. Entretanto, alguns detalhes da teoria foram melhorados no decorrer do tempo, refinando o conhecimento sobre a recordação histórica da evolução. Por exemplo, foram encontrados fósseis que nos mostram que não há uma evolução contínua no tempo. Na realidade, o que observamos são períodos nos quais ocorreram poucas mudanças e alguns picos de atividade evolucionária.

Um fato experimental nos mostra que existe um equilíbrio pontual cuja causa é desconhecida. Um modelo computacional sugerido por Bak e Sneppen [33] nos leva a esse tipo de comportamento. Esse modelo é baseado na ocorrência de coevolução, isto é, a evolução das espécies é afetada pela evolução das espécies com as quais elas interagem.

8.1 O modelo de coevolução

O modelo é executado em uma rede unidimensional de tamanho n, empregando-se condições de contornso periódicas. Esta rede representa o ecossistema em que cada sítio é uma espécie. O valor dado a cada sítio é um número aleatório. Os vizinhos próximos representam as espécies com as quais a espécie do sítio interage. O número aleatório representa a medida *fitness* da espécie, que faz o papel da barreira evolucionária que deve ser superada pela espécie. Essa barreira está relacionada com a quantidade de material genético que deve ser modificado para que uma espécie sofra uma mutação. Quanto maior a barreira, mais estável é a espécie. Assim, o processo de evolução por mutação e seleção natural ocorre em espécies menos desenvolvidas, isto é, naquelas em que a barreira é menor. A evolução de uma espécie, de um nível *fitness* para um outro, afeta as espécies com as quais ela interage. Por exemplo, a evolução de uma espécie pode afetar a cadeia alimentar, os sistemas presa predador de um ecossistema em um dado tempo. Assim, podemos representar a mutação pelo processo de encontrar o menor valor da rede e substituí-lo por um novo valor aleatório. Além disso, também temos de substituir os valores dos sítios vizinhos, direito e esquerdo. Na subseção a seguir, vamos apresentar o algoritmo que simula esse processo.

8.1.1 Algoritmo de coevolução

O algoritmo que gera o processo de coevolução pode ser dividido em quatro passos, como é descrito abaixo:

- **Primeiro passo**: criamos uma rede unidimensional de tamanho n e sorteamos um valor aleatório para cada sítio da rede. Também criamos uma lista vazia que chamaremos de lista das espécies menos evoluídas;
- **Segundo passo**: determinamos o sítio que possui o menor valor;
- **Terceiro passo**: colocamos a localização desse sítio na lista das espécies menos evoluídas. Além disso, substituímos o valor do sítio selecionado e dos seus primeiros vizinhos por novos números aleatórios;
- **Quarto passo**: repetimos essa sequência até uma quantidade desejada.

8.2 Resultados obtidos pelo modelo de coevolução

Utilizando o algoritmo da seção anterior, construímos o programa *DarwinianEvolution*, com o qual fizemos uma simulação de evolução para um ecossistema com 200 espécies envolvendo 8000 gerações, as quais olharemos apenas para as últimas 6000, como mostrado na figura 7.1, na qual podemos notar um comportamento não aleatório.

Um outro resultado interessante que mostra esse tipo de comportamento é o gráfico da frequência de mutações da figura 7.2. Algumas espécies não apresentam um número grande de evoluções, enquanto outras passam por grandes mudanças. De acordo com o modelo de coevolução, espécies que interagem entre si tendem a se envolver em uma moda intermitente (equilíbrio pontual). Além disso, espécies que se envolveram com uma mutação recente possuem mais chances de sofrer uma nova mutação do que espécies que estão imutáveis por longos períodos.

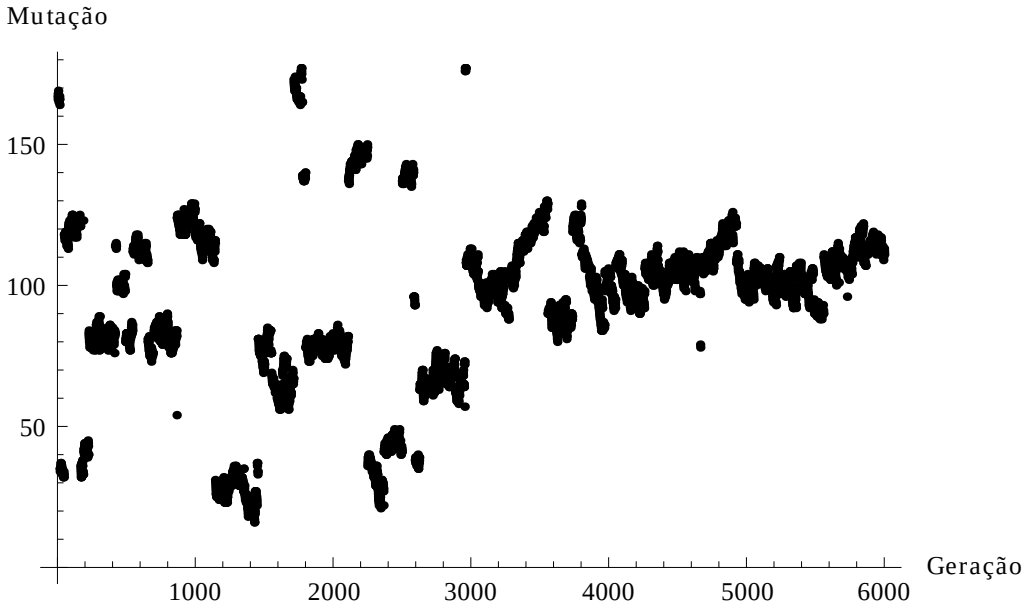

Figura 8.1: Representação gráfica de uma evolução envolvendo 200 espécies que interagem entre si.

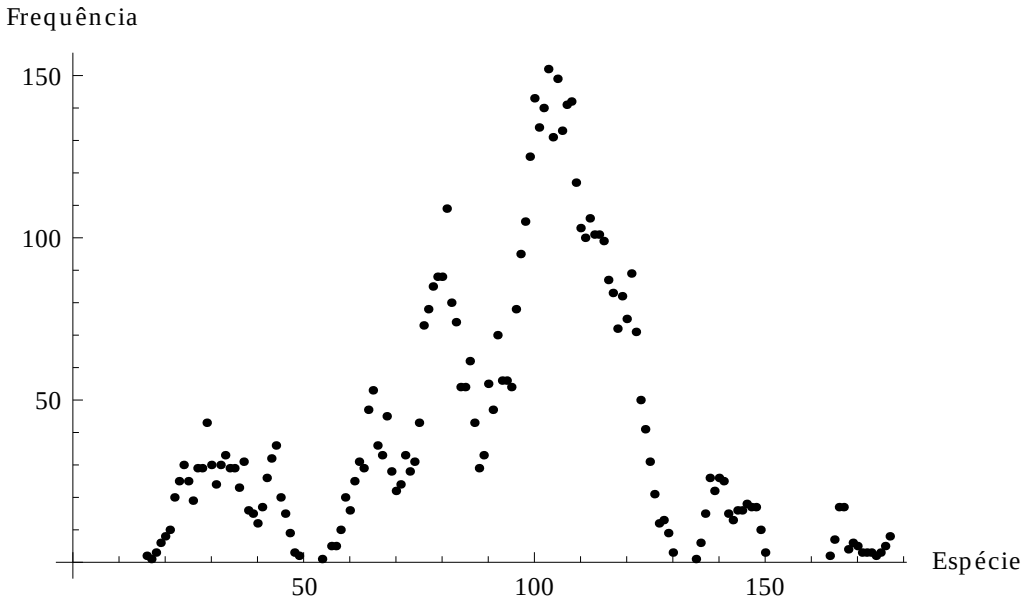

Figura 8.2: Representação gráfica da frequência de mutações por espécie.

Capítulo 9

Conclusões

Neste trabalho, em uma visão geral, ilustramos que a utilização de modelos computacionais no estudo de sistemas complexos é uma ferramenta poderosa e que tem dado muitos frutos à ciência. Por outro lado, especificamente nos modelos aqui discutidos, notamos que existem ramificações nas diversas áreas do conhecimento, muitas delas ainda não estudadas suficientemente. Por essa razão, acreditamos que este trabalho é uma porta de entrada para o estudo de problemas ainda em aberto da ciência atual.

Em relação aos modelos estudados, iremos descrever as principais conclusões a seguir. Por meio do modelo de caminhada aleatória, obtivemos, em nossas simulações, o valor do expoente ν, da relação $< r^2 > \propto n^{\nu}$, como sendo aproximadamente um, valor bem conhecido na literatura.

No modelo de caminhada autoexcludente, discutimos dois algoritmos que tentam reproduzir um mesmo processo. Entretanto, notamos que o algoritmo *Slithering Snake* possui algumas limitações, enquanto o algoritmo pivô mostrou-se mais eficiente. Ainda, obtivemos o expoente ν, da relação $< r^2 > \propto n^{\nu}$ de uma caminhada autoexcludente, como sendo aproximadamente $1,5$, o que sugere uma correlação entre os passos, como era de se esperar para esse tipo de caminhada. Por fim, também verificamos que a dimensão fractal era próxima de $4/3$.

Também calculamos a dimensão fractal para uma agregação por difusão limitada e concluímos que ela é negativa, o que significa que a estrutura do agregado é mais densa no centro e menos densa nas periferias. Além disso, obtivemos representanções gráficas da agregação por difusão limitada e da deposição balística que podem ser comparadas com vários processos que ocorrem na natureza.

De acordo com os modelos de percolação aleatória e percolação por invasão, conseguimos descrever um espalhamento sem usar os métodos tradicionais da física (resolver equações, por exemplo). Entretanto, o resultado obtido não é uma equação que descreve o processo, mas sim dados numéricos que representam a evolução do espalhamento.

Em relação ao modelo de percolação de aglomerados, conseguimos construir um programa que pode nos mostrar como eles são formados aleatoriamente e que existe uma probabilidade p_c (limiar de percolação), em que o aglomerado se espalha por toda a rede.

Através do modelo de Ising, conseguimos simular o comportamento dos spins em uma rede quadrada. Com isso, foi possível observar que a magnetização tende a um valor constante rapidamente a partir de uma dada configuração inicial.

O modelo de evolução mostrou que a coevolução nos leva a um comportamento não

aleatório. Além disso, revelou que espécies que sofreram uma mutação recentemente possuem mais chance de evoluírem novamente e que, por outro lado, espécies que passaram por um longo período sem mutações dificilmente evoluem.

Referências Bibliográficas

[1] R.J. Gaylord, P. R. Wellin, *Computer Simulations with Mathematica: Explorations in Complex Physical and Biological Systems* (Springer, New York, 1994).

[2] N. Boccara, *Modeling Complex Systems* (Springer, New York, 2010).

[3] R. Brown, *A brief account of microscopical observations made on the particles contained in the pollen of plants*, Philosophical Magazine **4**,161-173 (1828).

[4] A. Einstein, *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen*, Annalen der Physik **322**, 549-560 (1905).

[5] C.R. Nelson e C.R. Plosser, *Trends and random walks in macroeconmic time series: Some evidence and implications*, Journal of Monetary Economics **10**, 139-162 (1982).

[6] R.N. Mantegna e H.E. Stanley, *Introduction to Econophysics: Correlations and Complexity in Finance* (Cambridge University Press, Cambridge, 2000).

[7] M. Kröger, *Models for polymeric and anisotropic liquids* (Springer, Berlin, 2005)

[8] N. Madras e S. Gordon, *The Self-Avoiding Walk* (Birkhäuser, Boston, 1992).

[9] C. Scherer, *Métodos Computacionais da Física* (Livraria da Física, São Paulo, 2005).

[10] F. Reif, *Fundamentals of Statistical and Thermal Physics* (McGraw-Hill, New York, 1965).

[11] S.R.A. Salinas, *Introdução à Física Estatística* (EDUSP, São Paulo, 1997).

[12] K. Binder, *Monte Carlo and Molecular Dynamics Simulations in Polymer Science* (Oxford University Press, Oxford, 1995).

[13] T. Vicsek, *Fractal Growth Phenomena, 2nd Edition* (World Scientific Publishers, Singapore, 1992).

[14] T.A. Witten e L.M. Sander, *Diffusion-Limited Aggregation*, Physical Review B **27**, 5686-5697 (1983).

[15] M. Kolb, B. Botet e B. Jullie, *Scaling of Kinetically Growing Cluster*, Physical Review Letters **51**, 1123-1126 (1983).

[16] Z. Zhang e M.G. Lagally, *Atomistic Processes in the Early Stages of Thin-Film Growth*, Science **276** (5311), 377-383 (1997).

[17] M. Eden, *A Two-Dimensional Growth Process*, em *The Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, Volume **3**, 223-239 (University of California Press, California, 1961).

[18] H.J. Hermann, *Geometrical Cluster Growth Models and Kinetic Gelation*, Physics Reports **136**, 153-227 (1986).

[19] J.L. Cardy e P. Grassberger, *Epidemic models and percolation*, Journal of Physics A: Mathematical and General **18**, L267-L271 (1985).

[20] B. Pittel, *On Spreading a Rumor*, SIAM Journal on Applied Mathematics **47**, 213-223 (1987).

[21] B. Berkowitz e I. Balberg, *Percolation Theory and its Applications on Groundwater Hydrology*, Water Resources Research **29**, 775-794 (1993).

[22] M.J. Blunt, *Flow in porous media - pore-network models and multiphase flow*, Current Opinion in Colloid and Interface Science **6**, 197-207 (2001).

[23] M. Sahimi, *Applications Of Percolation Theory* (Taylor and Francis, London, 1994).

[24] L.L. Hench e J.K. Wes, *The Sol-Gel Process*, Chemical Review **90**, 33-72 (1990).

[25] P.G. DeGennes, *Percolation: A New Unifying Concept*, La Recherche **7**, 919 (1980).

[26] D. Stauffer e A. Aharony, *Introduction To Percolation Theory* (Taylor and Francis, London, 1994).

[27] J. Hoshen e R. Kopelman, *Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm*, Physical Review B **14**, 3438-3445 (1976).

[28] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller e E. Teller, *Equation of State Calculations by Fast Computing Machines*, The Journal of Chemical Physics **21**, 1087-1092 (1953).

[29] E. Callen e D. Shapero, *A theory of social imitation*, Physics Today **27**, 23-28 (1974).

[30] M.E.J Newmann e G.T. Barkema, *Monte Carlo Methods in Satistical Physics* (Claredon Press, Oxford, 1999).

[31] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer* (Addison-Wesley, Redwood, 1991).

[32] C. Darwin, *On the Origin of Species by Means of Natural Selection or Preservation of Favoured Races in the Struggle for Life* (John Murray, London, 1859).

[33] P. Bak e K. Sneppen, *Punctuated Equilibrium and Criticality in a Simple Model of Evolution*, Physical Review Letters **24**, 4083-4086 (1993).

[34] D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd edition* (Addison-Wesley, Boston, 1998).

Apêndice A

Números aleatórios

Uma noção geral sobre o que é um número aleatório todos nós conhecemos. Por exemplo, quando jogamos um dado é imprevisível saber a face, ou seja, o número que cairá para cima. Da mesma forma, após algumas jogadas, não saberíamos dizer qual seria o próximo valor. Então, sempre que pensamos em números aleatórios, temos a ideia de escolher um número qualquer cuja escolha independa de qualquer fator e de que a probabilidade de escolher ele é igual a de escolher qualquer outro. Entretanto, expressar tal número matematicamente não é uma tarefa tão fácil.

Na realidade, dificilmente imaginamos um número aleatório sozinho, sempre imaginamos uma sequência deles. Por esse ponto de vista, uma lista de números aleatórios consiste em uma sequência em que cada elemento é independente dos outros elementos da lista. Por exemplo, uma sequência de números tal como

$$\{1, 1, 2, 3, 5, 8, ...\} \tag{A.1}$$

é não aleatória, pois o elemento seguinte da lista é a soma dos dois elementos anteriores. Por outro lado, uma sequência como esta,

$$\{46, 35, 29, 61, 30, 100, 100, 21, 36, 85\}, \tag{A.2}$$

parece, em um sentido informal, ser aleatória.

Computadores podem gerar números de acordo com algum algoritmo prescrito que, em muitos aspectos, assemelham-se a números aleatórios. No entanto, tal sequência é determinística, pois, usando a mesma configuração inicial, podemos obter a mesma sequência. Por exemplo, execute várias vezes o seguinte comando no software *Mathematica*:

```
SeedRandom[4]
Table[Random[], {5}]
```

o resultado é sempre o mesmo:

```
{0.672212, 0.381237, 0.664015, 0.808308, 0.64736}
```

Como podemos observar, a sequência gerada é sempre a mesma, pois fixamos a condição inicial (SeedRandom[4]). Por esta razão, é comum chamar tais números como *números pseudo aleatórios*, embora, muitas vezes eles são chamados de números aleatórios.

Todas as linguagens de computadores utilizam geradores de números aleatórios para produzir sequências de números a partir de alguma operação repetida prescrita por algum algoritmo. Esta sequência, como em nosso exemplo, começa por uma semente e, aplicando o algoritmo, vamos obtendo novos números.

Um dos algoritmos que gera números aleatórios mais conhecidos é o *método congruente linear*. Começamos com uma semente x_0 e a sequência de números é determinada como segue:

$$x_n = (ax_{n-1} + c)(modm), \tag{A.3}$$

em que a é referido como *multiplicador*, c é chamado *incremento* e m é denominado *modulo* [1]. Apesar desse método gerar sequências de números aleatórios excelentes, podemos verificar que ele pode gerar algumas sequências periódicas. Por isso, devemos escolher corretamente os parâmetros a fim de obter sequências melhores. Uma primeira forma de testar se uma sequência de números aleatórios é boa seria verificando a frequência com que os números aparecem na lista, pois, para uma lista grande, esperaríamos que todos os números sorteados aparecessem, aproximadamente, com a mesma frequência, pois todos deveriam ser equiprováveis. Um tratamento completo sobre esse método pode ser visto na referência [34].

Apesar de geradores como esse criarem listas determinísticas, eles são muito úteis para nossos propósitos. Geradores de números aleatórios com diferentes distribuições de probabilidade são de grande uso e, por exemplo, no *Mathematica*, existem vários algoritmos que geram tais números com uma distribuição arbitrária, como pode ser visto na referência [1].

Apêndice B

Programas

Todos os programas usados em nossas simulações foram feitos na linguagem do *Mathematica*.

B.1 Caminhada aleatória

Caminhada aleatória em uma dimensão: O *Walk1D* gera uma lista que começa em zero e, a cada incremento, soma-se o valor anterior mais um número aleatório (+1 ou -1). O número n representa a quantidade de passos tomados.

```
Walk1D[n_] := FoldList[Plus, 0, Table[(-1)^Random[Integer], {n}]]
```

Caminhada aleatória em duas dimensões: o *Walk2D* gera uma lista de pares ordenados que começa no ponto $\{0, 0\}$ e, a cada incremento, soma-se o valor anterior mais um par aleatório. O número n representa a quantidade de passos tomados.

```
Walk2D[n_] :=
 FoldList[Plus, {0, 0}, {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}[[
   Table[Random[Integer, {1, 4}], {n}]]]
```

Valor médio do quadrado da distância de ponta a ponta: o *MeanSquareDistance* calcula o valor médio de r^2 (de acordo com a subseção 2.3.1), fazendo várias réplicas de uma caminhada aleatória em uma rede bidimensional. O número n representa a quantidade de passos tomados e m o número de réplicas.

```
MeanSquareDistance[n_Integer, m_Integer] :=
 Module[{Walk2D},
  Walk2D[s_] :=
   FoldList[
    Plus, {0, 0}, {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}[[
     Table[Random[Integer, {1, 4}], {s}]]];
  N[Sum[Apply[Plus, Last[Walk2D[n]]^2], {m}]/m]]
```

Valor médio do quadrado do raio de giração: o *MeanSquareRadiusGyration* calcula o valor médio de R_g^2 (de acordo com a subseção 2.3.2), fazendo várias réplicas de uma caminhada aleatória em uma rede bidimensional. O número n representa a quantidade de passos tomados e m o número de réplicas.

```
MeanSquareRadiusGyration[m_Integer, n_Integer] :=
 Module[{squareRadiusGyration},
  squareRadiusGyration[s_Integer] :=
   Module[{locs, cm, choices = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}},
    locs = FoldList[Plus, {0, 0},
      choices[[
       Table[Random[
         Integer, {1, 4}], {s}]]]];
         cm = N[Apply[Plus, locs]/(s +1)];
         Apply[Plus,
              Flatten[(Transpose[locs] - cm)^2]]/(s +1)
   ];
  N[Sum[squareRadiusGyration[n], {m}]/m]]
```

Visualizando uma caminhada aleatória: o *ShowWalk2D* lê os dados gerados pelo programa *Walk2D* e cria linhas que ligam um passo a outro da caminhada aleatória gerada.

```
ShowWalk2D[coords_, opts___] :=
 Show[Graphics[Line[coords], opts, AspectRatio -> Automatic]]
```

B.2 Caminhada autoexcludente

O cálculo do valor médio do quadrado da distância de uma caminhada autoexcludente é feito pelo programa *SlitheringSAW*, que produz caminhadas autoexcludentes de n-passos com m réplicas, para que, assim, possa ser calculado o valor de $< r^2 >$.

```
SlitheringSAW[n_, m_] := Module[{squaredist, snake},
  squaredist = n^2;
  snake = Module[{newpt, path,
     choice = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}},
     newpt = Last[#] + choice[[Random[Integer, {1, 4}]]];
  If[MemberQ[Rest[#], newpt],
       path = Reverse[#],
       path = Join[Rest[#], {newpt}]];
      squaredist += Apply[Plus, (First[path] - Last[path])^2]; path] &;
    Nest[snake, Table[{i, 0}, {i, 0, n}], m];
    N[squaredist/(m+1)]]
```

Valor médio do quadrado da distância de uma caminhada autoexcludente: o *Pivot2DSAW* tem a mesma função do programa *SlitheringSAW*. Porém, este programa possui algumas melhorias em relação ao anterior: ergodicidade e rapidez na simulação para obtermos grandes mudanças nas formas das caminhadas.

```
Pivot2DSAW[n_Integer, m_Integer] :=

 Module[{squaredist, twistAndShout},
  squaredist = n^2;
  twistAndShout =
   (Module[{ball, fixsec, movesec, rotchoice,
       newsec, newconfig,
       rot = {{{0, -1}, {1, 0}},
         {{0, 1}, {-1, 0}},
         {{-1, 0}, {0, -1}}}},
      ball = Random[Integer, {1, n - 1}];
      fixsec = Take[#, ball];
      movesec = Take[#, ball - (n + 1)];
      rotchoice = rot[[Random[Integer, {1, 3}]]];
      newsec = Map[Function[y, #[[ball]] +
          (y - #[[ball]]).rotchoice],
        movesec];
      If[Intersection[newsec, fixsec] == {},
       newconfig = Join[fixsec, newsec];
       squaredist += Apply[Plus, Last[newconfig]^2];
```

```
      newconfig,
      squaredist += Apply[Plus, Last[#]^2];
      #]]) &;
  Nest[twistAndShout, Table[{j, 0}, {j, 0, n}], m];
  N[squaredist/(m + 1)]]
```

B.3 Acreção

Gerando uma agregação por difusão limitada: o programa *DLA* gera uma lista das localizações das partículas após um processo de agregação. O número s define o raio da circunferência que delimita a região onde podemos iniciar uma caminhada e o número m define a quantidade de sítios que devem ser ocupados para que o processo seja finalizado.

```
DLA[s_Integer, m_Integer] :=
 Module[{loc, rad, particleCount = 0,
   stepChoices = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}}},
  occupiedSites = {{0, 0}};
  While[Length[occupiedSites] < m,
   ++particleCount;
   rad = Max[Abs[occupiedSites]] + s;
   loc = FixedPoint[
     (# + stepChoices[[Random[Integer, {1, 4}]]]) &,
     Round[rad {Cos[#], Sin[#]}] &[Random[Real,
       {0, N[2 Pi]}]],
     SameTest -> (Apply[Plus, (#2)^2] > (rad + s)^2 ||

         Intersection[occupiedSites,
           Map[Function[y, y + #2],
            stepChoices]] != {} &)
     ];
   If[Apply[Plus, loc^2] < rad^2,
    occupiedSites = Join[occupiedSites, {loc}]]
   ];
  Print["O número de partículas lançadas foi ",
   particleCount, "."];
  occupiedSites]
```

Visualizando uma DLA: os dois programas abaixo reproduzem graficamente uma DLA. No primeiro, *ShowDLA*, basta inserir a lista gerada pelo programa *DLA*, enquanto que, no segundo, *ShowDLA2*, inserimos a lista e, separado por uma vírgula, inserimos o comando *Background* $\rightarrow$ *GrayLevel[0.8]* para mudar a cor do fundo da imagem. A seguir, temos o código-fonte dos dois programas:

```
ShowDLA[sites_, opts___] := Module[{structure, s = Length[sites]},
  structure = {};
  Map[(AppendTo[structure,
      {GrayLevel[.99 #/s],
       Rectangle[sites[[#]] - {0.5, 0.5},
        sites[[#]] + {0.5, 0.5}],
       White,
       Text[#, sites[[#]]]}]) &,
   Range[s]];
  Show[Graphics[structure],
   opts,
   Axes -> None,
   AspectRatio -> Automatic,
   PlotRange -> All]
  ]

ShowDLA2[sites_, opts___Rule] :=
  Module[{len = Length[sites], points, colors},
  points = Map[Point, sites];
  colors = Map[GrayLevel, Range[len]/len];
```

```
  Show[Graphics[{PointSize[1/Sqrt[len]],
     Transpose[{colors, points}]},
    opts, Axes -> None,
    AspectRatio -> Automatic,
    PlotRange -> All]]]
```

Dimensão fractal de uma DLA: o programa *FractalDimension* calcula o valor da dimensão fractal para uma dada DLA.

```
FractalDimension[occupiedSites_List] :=
 Module[{occSiteDensity, fractalDataList, fractaldim},
  occSiteDensity[t_Integer] :=

   N[Count[occupiedSites, {x_?(Abs[#] <= t &),
       y_?(Abs[#] <= t &)}]]/(2 t + 1)^2;
  fractalDataList = Table[{2 s, occSiteDensity[s]},
    {s, Max[Abs[occupiedSites]]}];
  fractaldim = Fit[N[Log[fractalDataList]],
    {1, x}, x];
  Print["A dimensão fractal da DLA é ", Coefficient[fractaldim, x],
   "."];]
```

Construção de uma deposição balística: por meio do programa *MolecularDeposition*, simulamos uma deposição com n partículas iniciais e repetimos o processo até obtermos t partículas depositadas.

```
MolecularDeposition[n_, t_] :=
 Module[{init, newLat, nnColumns,
   depositRow, emitLayer},
  init = Transpose[Table[{0, 1}, {n}]];
  newLat = (depositColumn = Random[Integer, {1, n}];
     nnColumns =
      Transpose[#][[{depositColumn - 1 /.
            0 -> n,
          depositColumn,
          depositColumn + 1 /.
           (n + 1) -> 1}]] &;
     depositRow =
      Min[Flatten[Map[Function[y, First[Position[y, 1]]],
           nnColumns[#]]] - {0, 1, 0}] &;
     ReplacePart[#, 1, {depositRow[#], depositColumn}]) &;
  emitLayer = If[#[[1]] != Table[0, {n}],
     Prepend[#, Table[0, {n}]], #] &;
  Nest[emitLayer[newLat[#]] &, init, t]
  ]
```

Visualização de uma deposição: a partir do resultado obtido pelo programa *MolecularDeposition*, podemos visualizar a deposição usando o programa *ShowDeposition*.

```
ShowDeposition[sites_, opts___] :=
  ListDensityPlot[Reverse[sites /. {0 -> 1, 1 -> 0}],
  opts,
  AspectRatio -> Automatic,
  Mesh -> False,
  FrameTicks -> None,
  Frame -> None
  ]
```

B.4 Fenômenos de espalhamento

Percolação aleatória: o programa *Epidemic* simula a disseminação de doenças por meio do modelo de percolação aleatória. Os valores de entrada n e p representam, respectivamente, o número de sítios ocupados (contaminados) e a probabilidade de um sítio se juntar aos outros do aglomerado.

```
Epidemic[n_, p_] :=
 Module[{choices = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}},
   reject, pickAndChoose, select, newPers},
  reject = {};
  pickAndChoose :=
   (select = #[[2,
       Random[Integer, {1, Length[#[[2]]]}]]];
     If[Random[] <= p,
      newPers =
       Complement[Union[Map[Function[y, y + select],
          choices], #[[2]]],
        {select}, #[[1]], reject];
      {Join[#[[1]], {select}], newPers},
      reject = Join[reject, {select}];
      {#[[1]], Complement[#[[2]], {select}]}]) &;
  FixedPoint[pickAndChoose, {{{0, 0}}, choices}, n,
    SameTest -> (#2[[2]] == {} ||
       Length[#2[[1]]] == n &)][[1]]
  ]
```

Modelo de Eden: quando temos o caso particular em que $p = 1$, reduzimos o modelo de percolação aleatória ao modelo de Eden. O programa *Eden* é mais rápido devido ao fato de pular algumas etapas do processo de percolação aleatória, aquelas relacionadas à probabilidade p. Temos abaixo o código fonte do programa, sendo que o valor de entrada n é o número de sítios ocupados que serão ocupados ao fim do processo.

```
Eden[n_] := Module[{choices = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}},
    pickAndChoose, select, cluster, newPers},
  pickAndChoose :=
   (select = #[[2,
       Random[Integer, {1, Length[#[[2]]]}] ]];
     newPers =
      Complement[Union[Map[Function[y, y + select],
          choices], #[[2]]],
        {select}, #[[1]]];
       {Join[#[[1]], {select}], newPers}) &;
  cluster = Nest[pickAndChoose,
     {{{0, 0}}, choices}, n][[1]]
  ]
```

Percolação por invasão: o programa *Invasion* simula o fluxo de um fluido através de um meio poroso. O valor de entrada n representa o tamanho que o aglomerado deve alcançar ao fim do processo.

```
Invasion[n_Integer] :=
 Module[{pickAndChoose, nn, newnn, newpers, newPerLis,
   choices = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}}},
  pickAndChoose :=
   (newcluSite = Sort[#[[2]]][[1, 2]];
     nn = Map[Function[y, y + newcluSite], choices];
     newnn = Complement[nn, #[[1]],
       Transpose[#[[2]]][[2]]];
     newpers = Transpose[{Table[Random[],
         {Length[newnn]}],
        newnn}];
     newPerLis = Join[DeleteCases[#[[2]],
        {_, newcluSite}], newpers];
     {Join[#[[1]], {newcluSite}], newPerLis}) &;
  Nest[pickAndChoose,
    {{{0, 0}}, Transpose[{Table[Random[], {4}],
       choices}]},
    n][[1]]
  ]
```

Visualização: a seguir, temos um programa *ShowSpread* que nos dá uma representação gráfica dos dados gerados pelos programas que simulam fenômenos de espalhamento. O valor de entrada é uma lista que pode ser gerada pelos programas *Epidemic, Invasion* e *Eden*.

```
ShowSpread[list_, opts___] :=
 Show[Graphics[{GrayLevel[0.],
    Map[(Rectangle[# - {0.5, 0.5},
        # + {0.5, 0.5}]) &, list]}],
  opts,
  AspectRatio -> 1,
  PlotRange -> Map[({Min[#], Max[#]}) &,
    Transpose[list]]]
```

B.5 Percolação de aglomerados

Percolação de sítios aleatórios: o programa *SitePercolation* cria uma matriz $m \times m$ que simula uma percolação de sítios aleatórios para um dado valor p.

```
SitePercolation[p_, m_Integer] :=
 Table[Floor[1 + p - Random[]], {m}, {m}]
```

Visualização de uma percolação de sítios aleatórios: o programa *DisplayPercolation* cria uma matriz $m \times m$ simulando uma percolação de sítios aleatótios para um dado valor p. Entretanto, o resultado obtido por esse programa é representado por uma imagem em que os quadrados em cor preta são os sítios ocupados e os quadrados em cinza são os sítios vazios.

```
DisplayPercolation[p_, m_Integer] :=
 Show[Graphics[
   RasterArray[
    Reverse[Table[Floor[1 + p - Random[]], {m}, {m}]] /.
     {1 ->
       Black,
      0 -> GrayLevel[0.8]}]],
  AspectRatio -> 1]
```

Nomeando os aglomerados: este programa nomeia os aglomerados de acordo com o algoritmo de Hoshen-Kopelman. O valor de entrada é a lista (matriz) gerada pelo programa *SitePercolation*.

```
ClusterLabel[r_List] :=
 Module[{uup := u[[q - 1, k]], uback := u[[q, k - 1]],
   ulup := ul[[q - 1, k]], ulback := ul[[q, k - 1]]},
  AddZeros[w_] :=
   Prepend[w, Table[0, {Length[w[[1]]]}]];
  u = Transpose[AddZeros[Transpose[AddZeros[r]]]];
  ul = u /. 1 -> 0;
  ulp = {};
  Do[
   Which[u[[q, k]] == 1 && uup == 1 &&
     uback == 1 &&
     ulup =!= ulback,
    ul[[q, k]] =
     Min[ulp[[ulback]], ulp[[ulup]]];
    ulp[[Max[ulp[[ulback]], ulp[[ulup]]]]] =
     Min[ulp[[ulback]], ulp[[ulup]]],
    u[[q, k]] == 1 && uup == 1,
    ul[[q, k]] = ulup,
    u[[q, k]] == 1 && uback == 1,
    ul[[q, k]] = ulback,
```

```
    u[[q, k]] == 1,
    ul[[q, k]] = Max[ul] + 1;
    AppendTo[ulp, Max[ul]]
    ],
   {q, 2, Length[u]}, {k, 2, Length[u]}
   ];
  new = Range[Length[ulp]];
  relabelrules1 = Thread[new -> ulp];
  correctlabels = ulp //. relabelrules1;
  relabelrules2 = Thread[Union[correctlabels] ->
     Range[Length[Union[correctlabels]]]];
  ufinal = ul //. relabelrules1 /. relabelrules2
  ]
```

Visualização de uma percolação de aglomerados: o programa *ShowPercolation* cria uma representação gráfica para uma dada lista de aglomerados (matriz de aglomerados), obtida pelo programa *SitePercolation*.

```
ShowPercolation[cluster_List, opts___] := Module[{},
  Show[Graphics[RasterArray[
     Reverse[Map[GrayLevel[#/(Max[cluster])] &,
       cluster, {2}]]]],
   opts, AspectRatio -> 1]]
```

B.6 Modelo de Ising

Simulação do modelo de Ising: o programa *IsingMetropolis* simula a interação entre os spins em uma rede quadrada.

```
IsingMetropolis[n_, m_, B_, J_] :=

 Module[{energydiff, initconfig, flip, flipLis, monteCarloStepLis},
  energydiff[ 1, 4] =  2 (B + 4 J);
  energydiff[ 1, 2] =  2 (B + 2 J);
  energydiff[ 1, 0] =  2 (B + 0 J);
  energydiff[ 1,-2] =  2 (B - 2 J);
  energydiff[ 1,-4] =  2 (B - 4 J);
  energydiff[-1, 4] = -2 (B + 4 J);
  energydiff[-1, 2] = -2 (B + 2 J);
  energydiff[-1, 0] = -2 (B + 0 J);
  energydiff[-1,-2] = -2 (B - 2 J);
  energydiff[-1,-4] = -2 (B - 4 J);
  initconfig=2Table[Random[Integer],{n},{n}]-1;
  flip =
   (lat = #;
     {i1, i2} = {Random[Integer, {1, n}], Random[Integer, {1, n}]};
     If[i1 == n, dn = 1, dn = i1 + 1];
     If[i2 == n, rt = 1, rt = i2 + 1];
     If[i1 == 1, up = n, up = i1 - 1];
     If[i2 == 1, lt = 1, lt = i2 - 1];
     nnvalsum =
      lat[[dn, i2]] + lat[[up, i2]] + lat[[i1, rt]] + lat[[i1, lt]];
     2 lat[[i1, i2]] (B + J nnvalsum);
     If[energydiff[lat[[i1, i2]], nnvalsum] < 0 !! Random[] <
       Exp[-energydiff[lat[[i1, i2]], nnvalsum]],
      lat[[i1, i2]] = -lat[[i1, i2]]; lat, lat]
     ) &;
  flipLis = NestList[flip, initconfig, m];
  monteCarloStepLis=flipLis[[Range[1,m,n^2]]]


  ]
```

Vizualização da relaxação magnética: o programa *ShowIsingMagnetization* faz o gráfico da magnetização versus os passos tomados em uma simulação de interação entre spins. O valor de entrada é os dados gerados pelo programa *IsingMetropolis*.

```
ShowIsingMagnetization[
  list_] :=
 Module[{n = Length[list[[1]]], longRangeOrderList},
  longRangeOrderList =
   Map[Abs[Apply[Plus, Flatten[#]]/n^2] &, list];
  ListPlot[longRangeOrderList,
   PlotJoined -> True,
   PlotLabel -> "Parâmetro de ordem de longo alcance",
   PlotRange -> All,
   PlotStyle -> {Black},
   AxesLabel -> {"Passo", "M"}]
  ]
```

B.7 Evolução Darwiniana

Simulação de coevolução: o programa abaixo simula a coevolução que ocorre devido à interação entre espécies de uma mesmo ecossistema. Os valores de entrada n e t representam o número de espécies e a quantidade de mutações desejadas respectivamente.

```
DarwinianEvolution[n_, t_] :=
 Module[{prebiotic, fitness, leastFitSites},
  leastFitSites = {};
  prebiotic = Table[Random[], {n}];
  fitness = Function[y, ReleaseHold[
     ReplacePart[y, Hold[Random[]],
      Join[# - 1 /. 0 -> n,
         AppendTo[leastFitSites, #];
         #,
         # + 1 /. (n + 1) -> 1] &[Position[y, Min[y]]]
      ]]];
  Nest[fitness, prebiotic, t];
  Flatten[leastFitSites]
  ]
```

Frequência de evolução das espécies: o programa *Frequency* calcula a frequência com que as espécies evoluem. O valor de entrada é o resultado do programa *DarwinianEvolution*.

```
Frequency[lis_] := Map[{#, Count[lis, #]} &, Union[lis]]
```