Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Modelagem Matemática e Computacional

AVANÇOS PARA O MODELO DE PERCOLAÇÃO COM DOIS FLUIDOS EM $\mathbb{Z}^2$

Agosto de 2023

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Modelagem Matemática e Computacional

AVANÇOS PARA O MODELO DE PERCOLAÇÃO COM DOIS FLUIDOS EM $\mathbb{Z}^2$

Ricardo Tavares

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, UFPB, da Universidade Federal da Paraíba, como parte dos requisitos necessários à obtenção do título de Mestre em Modelagem Matemática e Computacional.

Orientadores: Ana Flávia Uzeda dos Santos Macambira
Sérgio de Carvalho Bezerra

João Pessoa
Agosto de 2023

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **RICARDO NEVES TAVARES**, candidato ao título de Mestre em Modelagem Matemática e Computacional, realizada no dia 25 de agosto de 2023.

Aos vinte e cinco dias do mês de agosto do ano de dois mil e vinte e três, às 16h30, via videoconferência, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do discente RICARDO NEVES TAVARES, vinculado a Universidade Federal da Paraíba sob a matrícula nº 20211025874, candidato ao grau de Mestre em "*Modelagem Matemática e Computacional*", na linha de pesquisa *"Modelagem Probabilística"*, do Programa de Pós-Graduação em Modelagem Matemática e Computacional. A comissão examinadora foi composta pelos professores Ana Flávia Uzeda dos Santos Macambira, Orientadora e Presidente da Banca; Hugo Leonardo Davi de Souza Cavalcante, Examinador Interno ao Programa; Sérgio de Carvalho Bezerra, Examinador Interno ao Programa; e Iesus Carvalho Diniz, Examinador Externo à Instituição. Dando início aos trabalhos, a Presidente da Banca cumprimentou os presentes, comunicou a finalidade da reunião e passou a palavra ao candidato para que fizesse, oralmente, a exposição do trabalho de dissertação intitulado *"Avanços para o modelo de percolação com dois fluidos em Z^2"*. Concluída a exposição, o candidato foi arguido pela Banca Examinadora, que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Gean Paulo P. M. de Barros, secretário do Programa de Pós-Graduação em Modelagem Matemática e Computacional (PPGMMC), lavrei a presente ata, que vai assinada por mim e pelos membros da Banca Examinadora.

João Pessoa, 25 de agosto de 2023.

Gean Paulo Pereira Maurício de Barros
Secretário do PPGMMC
SIAPE 2326476

Profª. Drª. Ana Flávia U. dos Santos Macambira
Orientadora (PPGMMC)

Ana Flávia Uzeda dos Santos Macambira
SIAPE: 2459248

Prof. Dr. Hugo Leonardo Davi de Souza Cavalcante
Examinador Interno ao Programa (PPGMMC)

Prof. Dr. Sérgio de Carvalho Bezerra
Examinador Interno ao Programa (PPGMMC)

Prof. Dr. Iesus Carvalho Diniz
Examinador Externo à Instituição (UFRN)

Catalogação na publicação
Seção de Catalogação e Classificação

T231a Tavares, Ricardo Neves.
Avanços para o modelo de percolação com dois fluidos em Z^2 / Ricardo Neves Tavares. - João Pessoa, 2023.
72 f. : il.

Orientação: Ana Flavia Uzeda Dos Santos Macambira.
Coorientação: Sergio de Carvalho Bezerra.
Dissertação (Mestrado) - UFPB/CI.

1. Informática. 2. Percolação. 3. Probabilidade. 4. Física estatística. I. Macambira, Ana Flavia Uzeda dos Santos. II. Bezerra, Sergio de Carvalho. III. Título.

UFPB/BC CDU 004(043)

Elaborado por Larissa Silva Oliveira de Mesquita - CRB-15/746

Aos meus pais, minhas irmãs,
meus sobrinhos, minha esposa e
meus filhos

Agradecimentos

com imensa gratidão que compartilho este momento de conclusão da minha dissertação de mestrado. Esta jornada de descoberta e aprendizado foi repleta de desafios e conquistas, e não poderia ter chegado a este ponto sem o apoio inestimável das pessoas que estiveram ao meu lado.

Gostaria de expressar minha profunda gratidão aos meus orientadores, Sérgio De Carvalho Bezerra e Ana Flavia Uzeda Dos Santos Macambira. Sua orientação dedicada, conhecimento profundo e insights valiosos foram os pilares fundamentais que guiaram o meu caminho ao longo deste projeto. Suas orientações perspicazes não apenas moldaram minha pesquisa, mas também ampliaram minha compreensão do campo de estudo. Agradeço por investirem seu tempo, energia e expertise em minha jornada acadêmica.

À minha esposa Larissa, minha rocha, minha fonte inabalável de apoio e motivação, meu agradecimento é profundo e sincero. Seu amor, paciência e constante encorajamento foram a força propulsora que me impulsionou nos momentos de incerteza. Seu apoio incondicional tornou possível conciliar os desafios acadêmicos com as demandas diárias da vida.

Aos meus filhos Henry e Gael, minha inspiração inesgotável, agradeço por iluminarem meus dias e me lembrarem do propósito por trás de cada esforço. Os sorrisos constantes foram uma fonte de energia que me motivou a persistir mesmo nos momentos mais desafiadores.

Este marco não é apenas meu, mas é o resultado de um esforço coletivo de todos aqueles que acreditaram em mim e me apoiaram ao longo desta jornada. Saibam que cada palavra de incentivo, cada gesto de carinho e cada momento compartilhado contribuíram para a realização deste objetivo.

Mais uma vez, a todos vocês que fizeram parte desta jornada, o meu mais profundo e sincero agradecimento. Juntos, celebramos a realização deste feito significativo.

Resumo da Dissertação apresentada ao PPGMMC/CI/UFPB como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AVANÇOS PARA O MODELO DE PERCOLAÇÃO COM DOIS FLUIDOS EM $\mathbb{Z}^2$

Ricardo Tavares

Agosto/2023

Orientadores: Ana Flávia Uzeda dos Santos Macambira
Sérgio de Carvalho Bezerra

Programa: Modelagem Matemática e Computacional

O presente trabalho aborda avanços significativos no estudo de modelos de percolação com dois fluidos em um plano bidimensional $\mathbb{Z}^2$.

Um elemento crucial deste estudo é a criação de uma estrutura de dados inovadora, projetada para armazenar informações à medida que são realizados sorteios de elos que podem ser abertos ou fechados. Esses sorteios ocorrem com probabilidades associadas aos fluidos vermelho p, fluido azul q e ausência de fluido $(1 - p - q)$. O modelo é caracterizado por uma dependência local, considerando tanto a abertura de elos quanto a ausência de fluido.

Para garantir a viabilidade do fluxo dos fluidos, uma função foi desenvolvida para avaliar a possibilidade de passagem nas direções norte, sul, leste e oeste a partir de um ponto específico. Essa função verifica a presença de fluido na direção escolhida, contribuindo para uma análise precisa da percolação.

A estratégia de sorteio foi realizada em coordenadas cartesianas, seguindo um padrão de espiral que segmentou um quadrado em múltiplos níveis.

Esse estudo proporciona insights valiosos sobre a percolação de dois fluidos em um plano $\mathbb{Z}^2$, ampliando nosso entendimento das interações complexas entre os fluidos e a estrutura da rede. As contribuições metodológicas e os resultados alcançados têm implicações significativas não apenas no campo da física estatística, mas também em áreas interdisciplinares que envolvem a compreensão de sistemas dinâmicos e seus comportamentos emergentes.

Abstract of Dissertation presented to PPGMMC/CI/UFPB as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DISSERTAÇÃO MESTRADO

Ricardo Tavares

August/2023

Advisors: Ana Flávia Uzeda dos Santos Macambira
Sérgio de Carvalho Bezerra

Program: Computational Mathematical Modelling

The present work addresses significant advances in the study of two-fluid percolation models in a two-dimensional plane $\mathbb{Z}^2$. A crucial element of this study is the creation of an innovative data structure, designed to store information as links are drawn that can be opened or closed. These draws occur with probabilities associated with red fluid p, blue fluid q and no fluid $(1-p-q)$. The model is characterized by a local dependence, considering both the opening of links and the absence of fluid.

To ensure the viability of fluid flow, a function was developed to evaluate the possibility of passage in the north, south, east and west directions from a specific point. This function checks the presence of fluid in the chosen direction, contributing to an accurate analysis of percolation.

The draw strategy was carried out in Cartesian coordinates, following a spiral pattern that segmented a square into multiple levels.

This study provides valuable insights into the percolation of two fluids in a $\mathbb{Z}^2$ plane, expanding our understanding of the complex interactions between fluids and network structure. The methodological contributions and results achieved have significant implications not only in the field of statistical physics, but also in interdisciplinary areas that involve the understanding of dynamical systems and their emergent behaviors.

Sumário

Lista de Figuras

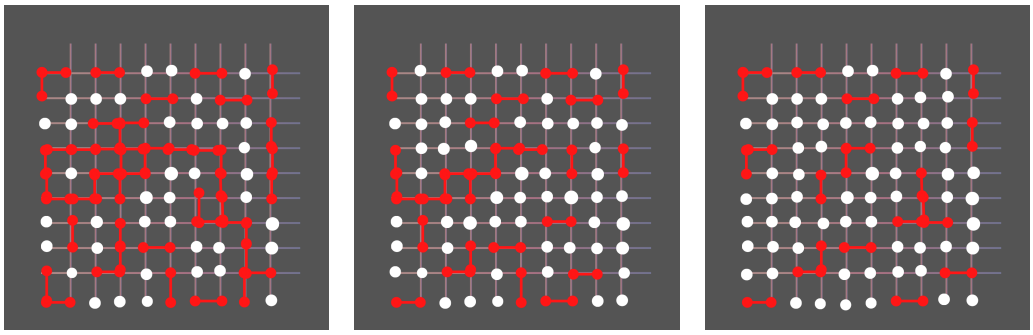
Capítulo 1

Introdução

Percolação é um problema físico que surgiu do estudo do fenômeno do transporte de um fluido através de um meio poroso, constituido de poros e canais microscópicos por onde passa o fluido. Os canais e poros que constituem o sistema podem estar abertos ou fechados à passagem do fluido, representando por exemplo, o modo pelo qual um líquido ou gás se infiltram através de uma rocha. O nosso objetivo neste trabalho é apresentar um algoritmo que resolva o problema da percolação de elos no conjunto $\mathbb{Z}^2$ para dois fluidos e para isto vamos descrever probabilisticamente uma distribuição de elos (arestas ou canais). Iniciaremos a abordagem com o modelo didático que trata da percolação com elos independentes. Vamos descrever a situação com apenas um fluido em que p representa a probabilidade de um elo estar aberto, e $(1-p)$ representa a probabilidade de estar fechado com $0 \leq p \leq 1$. A probabilidade de realização de cada sorteio de elos em um quadrado Q é dada por um produto de Bernoulli $p^{|A|}(1-p)^{|Q|-|A|}$ onde $|A|$ é o número de elos abertos e $|Q|$ é o número de elos do quadrado.

O primeiro modelo relacionado a este problema foi criado no século passado, sendo formulado mais especificamente no final da década de 1950 por Broadbent e Hammersley[1]. A primeira abordagem vem da suposição de que no plano bidimensional $\mathbb{Z}^2$ cada par de vértices vizinhos tenham um elo, tais que, cada elo está aberto com uma probabilidade p, fechado com probabilidade complementar e todos os elos são independentes entre si. Um fato é que para $p < p_c$ de modo que p_c é denotado valor *crítico* a probabilidade de percolação é nula, e para $p > p_c$ a probabilidade de percolação é positiva, caracterizando um sistema com transição de fase.

O problema de percolação possui diversas características e entre elas, podemos citar: renormalização(usado no cálculo de expoentes críticos para avaliar o comportamento de aglomerados infinitos, ou seja, $P^{co} \sim (p - p_c)^{\beta}$ onde β é o expoente crítico), transição de fase, fractais. Em nosso trabalho consideraremos apenas o parâmetro p. Existem valores de parâmetro que representam situações delicadas, como os pontos da fronteira entre duas fases distintas. Temos que, no exemplo



(a) (b) (c)

Figura 1.1: Modelos M_1 com: (a) p próximo a 1, (b) p próximo a $\frac{1}{2}$ e (c) p próximo a 0.

citado anteriormente o valor do ponto critico $p_c = \frac{1}{2}$.

Neste trabalho serão apresentados dois modelos a saber, M_1 e M_2, sendo o modelo M_1 apenas introdutório pois se trata do modelo mais simples de percolação, onde temos apenas um fluido e os elos serão abertos ou fechados respectivamente com probabilidade p e $1 - p$ de forma independentes uns dos outros, conforme a Figura 1.1 nas quais os pontos brancos indicam que não houve passagem de fluido e os vermelhos a passagem de fluido.

Abordamos o problema de percolação no modelo M_2, com dois tipos de fluidos, sendo um vermelho (V) e o outro azul (A). Neste modelo também temos dependencia espacial pois o fenômeno aberto ou fechado para cada elo depende das cores dos fluidos dos elos vizinhos . O estudo será desenvolvido através de uma simulação num quadrado Q centrado na origem, de modo que os pontos internos de coordenadas inteiras do quadrado, partindo da origem, serão percorridos no sentido anti-horario. Nós denotamos que se for escolhido por meio de um sorteio, o fluido vermelho (V) vai implicar em acender dois elos vermelhos de uma só vez e o fluido azul (A) acenderá um só elo. A chance de serem dois elos vermelhos é p a de ser um elo azul é q e a de não ter fluido (SF) é $1 - p - q$, sendo essas três probabilidades positivas e sua soma igual a 1. No modelo M_2 a ordem em que os sítios serão sorteados é determinística, partindo da origem, sorteando de maneira igualmente distribuida a direção e posteriormente sorteando V, A ou SF , partindo depois para a coordenada $(1, 0)$, seguindo assim sucessivamente sorteando as direções do espiral e sorteando os fluidos.

O principal objetivo deste trabalho é estabelecer uma estrutura de dados para a simulação dos modelos M_1 e M_2, bem como investigar as mudanças de fases para os dois modelos.

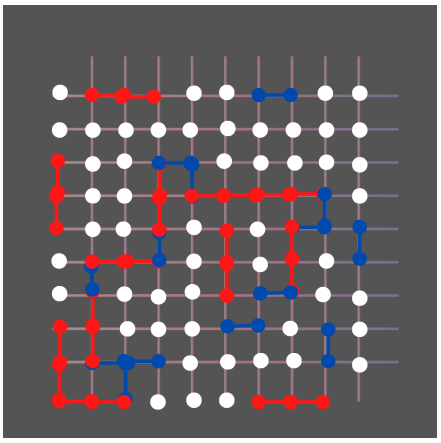


Figura 1.2: Modelo de percolação com dois fluidos em uma rede bidimensional 10x10
.

1.1 Percolação Definições e resultados iniciais

Para começarmos nosso estudo vamos introduzir o modelo matemático utilizado por Broadbent e Hammersley[1], no qual propuseram um meio poroso como sendo um substrato sólido,em seu interior há um certo número de pequenos canais e poros em pontos escolhidos ao acaso. Tal modelo é o modelo bidimensional de uma rocha imersa em um líquido. A percolação acontece quando há um número de canais suficientemente grande estando assim interligados, pois assim o meio se tornará permeável à passagem do fluido e portanto, o centro da rocha receberá fluido também. Apresentamos na figura abaixo uma representação de um recorte longitudinal de uma rocha, na qual seus canais e poros são representados por um grafo

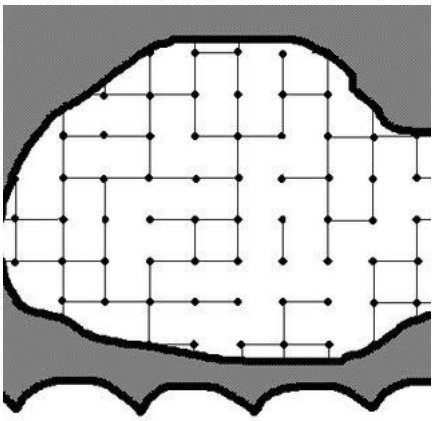


Figura 1.3: Modelo para uma rocha porosa bidimensional imersa em líquido [6]

1.1.1 Grafos

Um grafo é um par $G = (V, E)$, cujos elementos $V = \{v_1, v_2, ..., v_n\}$ e $E = \{e_1, e_2, ..., e_m\}$, são conjuntos enumeráveis chamados de vértices e arestas respectivamente, além disso o conjunto de vértices é não vazio. Grafo é uma estrutura usada para representar modelos onde existem relações entre objetos de uma certa coleção, muito utilizado para modelar diversas situações como por exemplo: rede de computadores, na biologia podemos usar para representar cadeia alimentar, na sociologia podemos representar redes sociais, na política as relações de poder, entre varias outras aplicações.

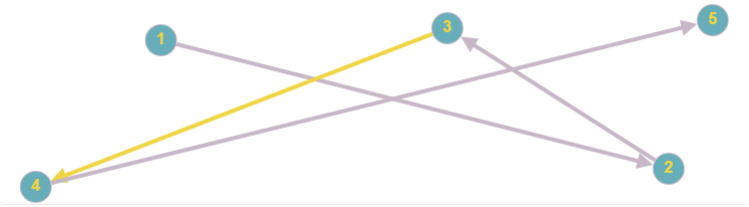


Figura 1.4: Modelo de grafo direcionado

1.1.2 Modelo probabílistico

Um modelo probabilístico é uma ferramenta matemática que ajuda a lidar com a incerteza ao representar eventos e resultados em termos de probabilidades. Assim dado o experimento e seus possíveis resultados, podemos definir a probabilidade de ocorrência dos mesmos à medida que o experimento se repete. Neste contexto a aleatoriedade é intríseca ao modelo, assim o mesmo experimento repetido por um número finito e conhecido de ensaios, exibe resultados diferentes, podendo-se obter apenas o padrão de regularidade nas ocorrências. Deste modo probabilidade, experimento aleatório, evento, espaço amostral, raciocinio dedutivo da probabilidade, raciocinio indutivo gerado a partir de uma estrutura e dados capazes de revelar o modelo, regularidade estocástica, são conceitos subjacentes a construção de um modelo probabilistíco. Para melhor compreensão, vamos definir alguns dos termos citados. Para isto usamos como base o livro do professor Sheldon Ross [7]

Definição 1.1.1 *Um experimento aleatório é aquele em que o resultado não pode ser previsto com certeza devido à natureza probabilística ou aleatória dos eventos envolvidos. Esses experimentos desempenham um papel fundamental na teoria das probabilidades e são amplamente usados em estatística, ciência e muitas outras disciplinas para modelar situações onde a incerteza é um elemento importante. Exemplo,no lançamento de um dado não viciado, o que podemos afirmar é qual a probabilidade de um certo número que está em uma das faces cair virado para cima.*

Definição 1.1.2 *Um espaço amostral Ω, é o conjunto de todos os resultados possíveis de um experimento aleatório.*

Definição 1.1.3 *Um evento é um conjunto formado pelos possíveis resultados de um experimento, assim temos que dado Ω um espaço amostral, enumerável, qualquer subconjunto A é um evento de Ω, além disso temos que A^c também um evento de Ω chamado de evento complementar, de modo que A^c ocorre se A não ocorrer.*

Definição 1.1.4 *Uma sigma-álgebra $\mathcal{M}$ é uma familia de subconjuntos/eventos de Ω, se satisfaz as seguintes propriedades:*

1. $\Omega \in \mathcal{M}$;
2. Se $A \in \mathcal{M} \Longrightarrow A^c \in \mathcal{M}$;
3. Se $A_1, A_2, ... \in \mathcal{M} \Longrightarrow \bigcup_{i=1}^{\infty} A_i \in \mathcal{M}$

Definição 1.1.5 *Para cada evento A elemento de Ω, uma função $\mathcal{P}$ a valores reais definida em $\mathcal{M}$ e que satisfaz os axiomas de Kolmogorov:*

1. $0 \leq P(A) \leq 1$
2. $\mathcal{P}(\Omega) = 1$
3. Para cada sequência de eventos mutuamente exclusivos $A_1, A_2, ... \in \mathcal{M}$,

$$\mathcal{P}\Big(\bigcup_{i=1}^{\infty} A_i\Big) = \sum_{i=1}^{\infty} P\Big(A_i\Big)$$

é chamada medida de probabilidade.

Definição 1.1.6 *Um espaço de probabilidade é dado pela tripla $(\Omega, \mathcal{M}, \mathcal{P})$*

Definição 1.1.7 *Uma variável aleatória X é uma função matemática que associa números reais (contradominio) aos resultados de um espaço amostral (dominio). Desta forma X é uma função mensurável em um espaço de probabilidade $(\Omega, \mathcal{M}, \mathcal{P})$ dada por:*

$$X : \Omega \longrightarrow \mathbb{R} \Longleftrightarrow \{\omega : X(\omega) \leq x\} \in \mathcal{M}, \forall\ x \in \mathbb{R}$$

As aplicações de modelos probabilístiscos possuem diversas finalidades, tais como: estimar, inferir, prever. Dentre os modelos probabilísticos mais comuns podemos citar os de variáveis discretas, como: Binomial que serve por exemplo para observar o número de componentes defeituosos em uma amostra de um grande número de peças, o número de caras em n lançamentos independentes de uma moeda

não-enviesada, o de Poisson que serve por exemplo para observar a desintegração de núcleos de substâncias radioativas, a quantidade de peças defeituos produzidas por hora em uma fábrica, e além desses podemos citar também o hipergeométrico, geométrico entre outros, dentre os de variáveis contínuas podemos citar: o modelo Gaussiano, o modelo uniforme, o qui-quadrado e muitos outros.

1.1.3 Processo Estocástico

Uma sequência de observações de um fenômeno é entendido como um processo, e sua principal forma de avaliação se dá pela observação no tempo, já a palavra estocástico significa aleatório, assim portanto um processo estocástico pode ser entendido como uma sequência cronológica de observações de um determinado fenômeno, de modo que sua evolução pode ser descrita por uma ou mais distribuições de probabilidades, e diferentes das soluções de uma EDO que possuem um único modo de evoluir, em um processo estocástico mesmo que se conheça a condição inicial, existem várias, por vezes infinitas direções nas quais o processo pode evoluir. Assim temos por definição que dado um espaço de probabilidade $(\Omega, \mathcal{M}, \mathcal{P})$, e para todo subconjunto $\mathcal{S}$ de Ω que pertence a $\mathcal{M}$, seja mensurável com relação a $\mathcal{P}$, onde Ω é o espaço amostral, $\mathcal{M}$ uma sigma-álgebra, $\mathcal{P}$ uma medida de probabilidade. Um processo estocástico é uma familia de variáveis aleatórias $(X_t)_{t \in T}$, em que T é chamado conjunto de instantes de tempo, onde o processo também pode ser representado como a função abaixo:

$$\begin{aligned} X :& T \times \Omega \to E \\ & (t, \omega) \to X_t(\omega). \end{aligned}$$

A cada instante de tempo $t \in T$ e $\omega \in \Omega$ tem como imagem um $e \in E$, onde E é chamado espaço de estados [9]. Os processos estocásticos tem uma aplicação muito vasta, por exemplo no mercado financeiro, que foi um dos principais motivadores da formalização dos estudo na área, que começou de forma mais rigorosa no início do século XX, onde é amplamente aceito que a evolução do preço de ativos financeiros pode ser modelado por um processo estocástico em tempo contínuo. Estes processos podem se dar em tempo discreto que são aqueles onde o valor da variável pode se alterar somente em intervalos pré-definidos de tempo (Ex: Cadeia de Markov), e processos de tempo contínuo onde o valor da variável está em constante mudança de forma aleatória seguindo alguma distribuição de probabilidades (Ex: Movimento Browniano, Flutuações do mercado de ações).

1.2 Objetivos

1.2.1 Gerais

Estudar o Comportamento assintótico de dois modelos de percolação, ambos com dependência espacial.

1.2.2 Específicos

- Construção de uma estrutura de dados para a simulação do modelo M_1
- Construção de uma estrutura de dados para a simulação de modelo M_2

Capítulo 2

Revisão Bibliográfica

2.0.1 Percolação - Caso Unidimensional

A capacidade de modelar e analisar o comportamento de redes sujeitas a elos abertos é fundamental para entendermos a resiliência, robustez e funcionamento de sistemas complexos do mundo real, visto que esses tipos de elos representam ligações que podem ser temporárias, frágeis ou sujeita a falhas.

Ao longo desta seção, iremos revisar os principais estudos realizados sobre percolação, examinando os fundamentos teóricos, as abordagens metodológicas utilizadas e os resultados obtidos. Serão explorados trabalhos clássicos e contemporâneos, bem como teses e artigos em periódicos cientifícos relevantes, a fim de oferecer uma visão abrangente do estado atual do conhecimento nessa área.

Em seu artigo intitulado "Introdução à teoria da percolação"dos professores Nelson Tanaka e Héliton Tavares [8], é feito a demonstração para o caso de percolação unidimensional, que descreve a propagação de um fluido como uma cadeia de sitios ou nós interconectados onde um aglomerado é caracterizado por um conjunto de sitios ocupados com os vizinhos dos extremos desse conjunto estando vazio, como mostra a figura a seguir:

Figura 2.1: Modelo de cluster unidimensional de tamanho 3 e tamanho 2

.

Usaremos a demonstração do artigo [8]. Consideremos inicialmente uma rede finita de tamanho L (L grande). A probabilidade de que haja um aglomerado de tamanho s é: $p^s(1-p)^2$, onde p é a probabilidade de um sítio qualquer estar ocupado e $(1-p)$ é a probabilidade de um sítio estar vazio, assim se um sítio a esquerda do aglomerado estiver vazio, ou um sítio a direita estiver vazio então não temos um aglomerado de tamanho infinito, por esse motivo $(1-p)^2$ indica a probabilidade de ter um sitio vazio no extremo da esquerda ou da direita. O número esperado

de aglomerados de tamanho s é : $Lp^s(1-p)^2$.Vamos definir o número esperado de aglomerados por sítio, n_s.Teremos então:

$$n_s = p^s(1-p)^2 \tag{2.1}$$

Para $p < 1$, o número de aglomerados vai a zero quando $s \to \infty$:

$$\lim_{s\to\infty} n_s = \lim_{s\to\infty} p^s(1-p)^2 = (1-p)^2 \lim_{s\to\infty} p^s = 0 \tag{2.2}$$

E também o número esperado de sítios vazios vai a infinito quando

$$L \to \infty = \lim_{L\to\infty} (1-p)L = (1-p) \lim_{L\to\infty} L = \infty \tag{2.3}$$

Assim, não teremos um aglomerado de tamanho infinito, ou seja, não haverá percolação se $p < 1$. Com isso concluimos que $P_c = 1$.

2.0.2 Modelo Teórico

Para representação do nosso problema, usaremos o estudo do artigo do Professor Luis Fontes[3]. Consideremos uma malha hipercúbica em d dimensões denotado por $(\mathbb{Z}^d, \mathbb{E}^d)$, onde $\mathbb{Z}^d$ é o conjunto de sitios da malha e

$$\mathbb{E}^d = \{(x,y)|x,y \in \mathbb{Z}^d : \|x-y\|_1 = 1\}$$

é o conjunto de elos vizinhos mais próximos.

Para cada elo de $\mathbb{E}^d$ iremos atribuir de forma aleatória o status de aberto ou fechado de modo que $\mathbb{X} := \left\{\mathbb{X}_e, e \in \mathbb{E}^d\right\}$ é uma familia de variáveis aleatórias independentes e identicamente distribuidas com distribuição comum de *Bernoulli* com parâmetro p, isto é:

$$P_p(\mathbb{X}_e = 1) = 1 - P_p(\mathbb{X}_e = 0) = p$$

Temos que para todo elo $e \in \mathbb{E}^d$, com p real variando entre 0 e 1, $P_{p,d}$ é a probabilidade associada a $\mathbb{X}$ com d indicando a dimensão e E_p é a esperança associada a esta probabilidade.

O espaço amostral do modelo é dado por Ω, a sigma-álgebra é $\mathcal{M}$, gerada pelos eventos que dependem apenas de elos em subconjuntos finitos de $\mathbb{E}^d$.

P_p ou ($P_{p,d}$, que é a outra forma de denotar), é a probabilidade produto em Ω, de modo que atribui peso p a 1 e $(1-p)$ a 0, e $\mathbb{X}_e(\omega) = \omega_e$ é a projeção na coordenada e, para todo $\omega \in \Omega$, e ainda se $\mathbb{X}_e = 1$, então temos que o elo e está aberto, e se $\mathbb{X}_e = 0$ então o elo e está fechado.

Temos ainda do artigo de [3], que um conjunto de elos de $\mathbb{E}^d$ $\{ e_1, e_2, ..., e_n\}$,

$n \geq 1$, onde $e_i = (x_i, y_i)$, $i = 1, 2, ..., n$, será dito um caminho se $x_1, x_2, ..., x_n$ forem distintos e $y_i = x_i + 1$, $i = 1, 2...(n-1)$, assim um caminho será dito aberto, se todos os seus elos estiverem abertos. Dois sítios da rede, estão conectados (notação: $x \leftrightarrow y$) se existir um caminho aberto de elos com $x_1 = x$ e $y_n = y$.

Um aglomerado é um conjunto de elos conectados, o qual denotaremos por $\mathcal{C}$, o aglomerado que parte da origem, e $\mathcal{C}_x$, o aglomerado do sítio x. Dessa maneira um dos objetivos do professor [3] é o de saber se aglomerados infinitos podem ocorrer com probabilidade positiva, como a constatação quase certamente de que em uma dimensão e com $p < 1$ não há aglomerados infinitos é trivial, o artigo se atém ao caso onde $d \geq 2$, onde é dada a função :

$$\theta(p) := P_p(|\mathcal{C}| = \infty) = 1 - \sum_{k=1}^{\infty} P_p(|\mathcal{C}| = k) \tag{2.4}$$

para calcular a distribuição da cardinalidade $|\mathcal{C}|$ do aglomerado $\mathcal{C}$ que é a mesma cardinalidade do aglomerado $\mathcal{C}_x$, por conta da invariância por translação de P_p

Teorema 2.0.1 *para $d \geq 2$, existe um valor crítico do parâmetro p, denominado p_c, no intervalo aberto $(0, 1)$ tal que*

$\theta(p) = 0$, *se* $p < p_c$

$\theta(p) > 0$, *se* $p > p_c$

Aqui não colocaremos a demonstração do teorema, para mais detalhes, procurar no trabalho citado.

2.0.3 Teoria da percolação de redes

A percolação é um problema estudado em física estatística que envolve a conectividade em redes aleatórias. A simulação de percolação é um abordagem comum para investigar propriedades de redes e entender como a conectividade afeta o comportamento do sistema. Neste cenário não é dificil de se deduzir que a teoria da percolação de redes é um assunto que vem sendo estudado nas mais diferentes perspectivas.

A teoria da percolação de redes utiliza uma série de conceitos e medidas para analisar as propriedades de clusters e entender como a conectividade se desenvolve em diferentes tipos de redes. Alguns dos conceitos mais importantes incluem a densidade de ocupação, que descreve a fração de nós ou elementos ocupados na rede, e a probabilidade de percolação, que é a probabilidade de que um determinado nó pertença ao cluster percolante. Dentro de uma ampla gama de modelos utilizados para se estudar a formação de clusters, os mais básicos são: O de sítios, que é o que considera uma rede onde cada nó pode estar ocupado ou vazio, e a formação de um cluster ocorre quando nós ocupados estão conectados, e o de vínculos, que considera

uma rede onde cada aresta ou vínculo pode estar ativo ou inativo, e a formação de um cluster ocorre quando vínculos ativos estão conectados.

No trabalho de dissertação do professor Gabriel Mendes [5] em seu último capítulo, podemos ver uma aplicação do estudo da teoria de percolação de redes, como tráfego veicular em redes complexas, onde é mostrado como o congestionamento cresce em uma rede de rodovias, quando este é submetido a um fluxo crescente de carros. Foram feitos dois modelos de simulação de tráfego veicular, o que queremos citar é um modelo análogo elétrico, no qual é utilizado resistores ôhmicos e não ôhmicos, segundo o autor neste modelo a medida que a pressão dos motoristas é aumentada, as ruas congestionam, como resistores queimam a medida que a diferença de potencial entre as extremidades ultrapassa um limiar. Nas palavras do autor "Este modelo trata o tráfego veicular como um sistema elétrico, considerando os veículos como elétrons, as ruas como fusíveis, o fluxo de carros como a corrente elétrica, a diferença de potencial entre dois pontos como a necessidade que um grupo de veículo tem em deslocar entre duas regiões, a condutância de um fusível como fluidez e por fim uma rua congestionada como um fusível queimado". Para isto a representação da rede é dada por um grafo direcionado $G = (N, S)$ onde os vértices $i \in N := 1, ..., n$ e cada ligação é um fusível com uma condutância ôhmica,

$$\sigma_{jk}^{l} = \begin{cases} v_{jk}^{\max}/l_{jk}, \text{ se } |V_j - V_k| \leq \Delta V_c \\ 0, \text{ se } |V_j - V_k| > \Delta V_c \end{cases}$$

"Onde ΔV_c é a diferença de potencial crítica, V_j é o potencial no sítio j, $v_{jk}^{\max}$ é a velocidade máxima permitida e l_{j_k} é o comprimento da rua entre os sítios j e k. Isto significa que o valor da condutância não varia com o aumento do potencial externo aplicado até a diferença de potencial crítica ΔV_c ser excedida, queimando o fusível, o que resulta num fusível com condutância nula"[5]. Segundo o autor a escolha do modelo acima se deu por conta de que, o fato dos motoristas desejarem economizar tempo em seus trajetos, acabam por optar por caminhos curtos e rodovias rápidas ao invés de ruas longas e lentas. Logo por esse motivo a condutância é inversamente proporcional ao tempo gasto pelo motorista ao se dirigir do cruzamento j ao i. Deste modo podemos notar que o caso do modelo citado acima se trata uma forma de percolação inversa.

A transição da percolação inversa é um fenômeno em que a adição de nós (vértices) ou arestas em um sistema complexo leva à fragmentação em vez de uma maior conectividade. Ao contrário da percolação convencional, em que a adição de elementos leva à formação de um cluster gigante. Em teoria de percolação de redes é comum denotar f como sendo uma fração de nós ativados ou seja vértices de uma rede quadrada por exemplo. Neste caso o valor do limite crítico, denotado por f_c,é o limiar de percolação, assim a percolação é inversa pois para $f < f_c$, temos um

componente gigante e pouco impacto na rede, para $f > f_c$, temos a formação de vários pequenos componentes isolados. O interesse nesse tipo de estudo é verificar a robustez de uma rede e qual o impacto das falhas dos nós, sendo que quanto mais robusta é a rede, maior é a capacidade do sistema resistir a perturbações e manter seu desempenho funcional mesmo diante de falhas ou ataques.Estudos teóricos como [4] têm mostrado que a transição de percolação inversa pode afetar a robustez de sistemas complexos. Um exemplo de aplicação pode ser em redes de transporte, por exemplo, a adição de novas rotas ou conexões pode resultar na fragmentação da rede em várias partes isoladas, comprometendo sua capacidade de transporte eficiente. A compreensão desses fenômenos tem implicações práticas significativas. Em redes de infraestrutura crítica, como fornecimento de energia,transporte e comunicação, entender esse processo de transição de fase inversa, pode auxiliar no planejamento e na tomada de decisões para garantir a resiliência desses sistemas. Além dos exemplo ja citados a análise de percolação inversa pode ser aplicada a redes sociais e sistemas de interação on line. Compreender como a adição ou remoção de conexões em uma rede social pode levar à fragmentação ou à resistência a pertubações ajuda a entender a propagação de informações e a formação de comunidade dentro desses sistemas.

2.0.4 Percolação e Deep Learning

O deep learning é uma abordagem revolucionária no campo da inteligência artificial, que se baseia em redes neurais artificiais profundas. Em seu livro sobre Deep Learning [2] o Autor Rogério Ferreira, nos traz que essa técnica permite que as máquinas aprendam e tomem decisões complexas, imitando o funcionamento do cerébro hunano, ou seja se trata de modelos computacionais bioinspirados. Ao contrário de métodos tradicionais de machine learning, o deep learning é capaz de extrair características e padrões diretamente dos dados brutos, eliminando a necessidade de uma prévia extração de recursos. O autor também traz em seu livro [2] os fundamentos de redes neurais artificiais, que é uma das bases essenciais do campo do deep learning. As redes neurais artificiais são compostas por um conjunto interconectado de unidades chamadas de neurônios artificiais ou nós. Cada neurônio recebe entradas ponderadas, realiza um cálculo e produz uma saída. O objetivo é simular o processamento de informações realizado pelo cérebro humano e utilizar esse paradigma para solucionar problemas complexos, assim uma rede neural é organizada em camadas, com uma camada de entrada, uma ou mais camadas intermediárias (camadas ocultas) e uma camada de saída. Cada camada é composta por neurônios interconectados,e as conexões entre eles são representadas por pesos. Além disso, são aplicadas funções de ativação nos neurônios para introduzir não - linearidade

e melhorar a capacidade de representação do modelo. As redes neurais artificiais têm sido amplamente aplicadas em diversas áreas, entre elas: visão computacional, processamento de linguagem natural, finanças, medicina, previsão do tempo, entre outras áreas, mostrando seu potencial em lidar com problemas complexos e não lineares.

Embora o deep learning não seja amplamente aplicado diretamente a problemas de percolação, existem abordagens relacionadas que podem fornecer insights valiosos para trabalhos futuros.

Uma aplicação possível é o uso de modelos generativos baseados em redes neurais, como as Redes Adversariais Generativas (GANs), que se trata de arquiteturas de redes neurais profundas compostas por duas redes colocadas uma contra a outra (daí o nome "adversárias"), para gerar redes aleatórias com propriedades específicas. Essas redes geradas podem ser aplicadas ao estudo de percolação e utilizadas para investigar a probabilidade de percolação ou ainda no caso de percolação de redes, para investigar a resitência da rede à falha.

Além disso, o deep learning pode ser empregado para extrair características relevantes das redes, como medidas de centralidade, distribuição de grau e agrupamento. Essas características podem ser usadas para avaliar a probabilidade de percolação em uma rede ou identificar regiões críticas.

Outra possibilidade é o uso de deep learning para otimizar a estrutura da rede com o bjetivo de evitar a percolação. Isso pode ser alcançado por meio de treinamento de um modelo para encontrar a topologia ideal que minimize a probabilidade de percolação ou maximize a resistência da rede a falhas.

O uso de deep learning em estudos de percolação apresenta um potencial significativo para avançar nosso entendimento sobre processos percolativos em sistemas complexos. Essa linha de pesquisa tem o potencial de contribuir para o desenvolvimento de materiais avançados, otimização de sistemas de transportes e outros campos relacionados à percolação.

Capítulo 3

Fundamentação Teórica

A fundamentação teórica tem um papel essencial neste trabalho, pois é ela que servira de alicerçamento intelectual para a compreensão aprofundada do tema em questão. Antes de começar a descrição dos modelos vamos pensar em uma situação hipotetica, onde veremos como acontece a percolação etapa por etapa.

3.0.1 Percolação Caso hipotético

Vamos considerar a fronteira da nossa malha todos os pontos que tenham abcissa ou ordenada igual a 3. Suponhamos que estamos na coordenada $(0, 0)$, onde realizamos o sorteio da direção. Vamos supor que seja sorteada a direção oeste primeiramente e no sorteio de fluido, seja sorteado o fluido vermelho, assim temos:

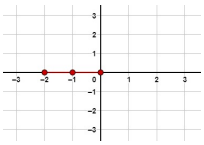


Figura 3.1: etapa 1

.

Agora passamos para a próxima coordenada que é $(1, 0)$, onde será realizado o próximo sorteio de maneira semelhante ao primeiro ponto. Suponhamos que seja sorteado, direção sul e fluido azul, assim atualizando a imagem teremos:

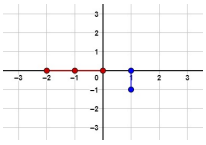
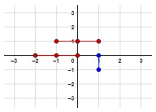


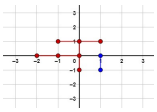
Figura 3.2: etapa 2

.

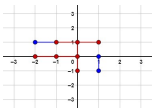
O sorteio é feito em coordenadas de um espiral, façamos agora alguns passos: coordenada $(1, 1)$, fluido vermelho, direção oeste



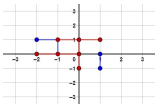
(3)



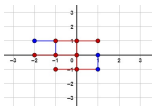
(4)



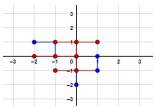
(5)



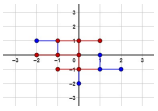
(6)



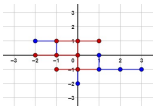
(7)



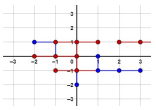
(8)



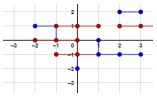
(9)



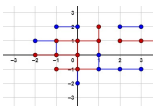
(10)



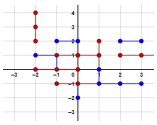
(12)



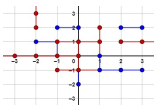
(13)



(15)



(16)



(18)

coordenada $(0, 1)$, fluido vermelho, direção sul
coordenada $(-1, 1)$, fluido azul, direção oeste
coordenada $(-1, 0)$, fluido azul, direção norte
coordenada $(-1, 1)$, fluido vermelho, direção leste
coordenada $(0, -1)$, fluido azul, direção sul
coordenada $(1, -1)$, fluido azul, direção leste
coordenada $(2, -1)$, fluido azul, direção leste
coordenada $(2, 0)$, sem fluido
coordenada $(2, 1)$, fluido vermelho, direção leste
coordenada $(2, 2)$, fluido azul, direção leste
coordenada $(1, 2)$, fluido, vermelho, direção sul
coordenada $(0, 2)$, sem fluido
coordenada $(-1, 2)$, fluido azul, direção leste
coordenada $(-2, 2)$, fluido vermelho, direção norte
coordenada $(-2, 1)$, sem fluido
coordenada $(-2, 0)$, fluido vermelho, direção oeste
Assim nesse caso vemos que a percolação acontece da coordenada $(0, 0)$ à $(-3, 0)$

3.0.2 Descrição dos Modelos M_1 e M_2

Nós começamos definindo uma família de quadrados Q_i com $i \in \mathbb{N}$, onde o quadrado Q_i, possui como dois de seus vértices $(-i, -i)$ e (i, i). Todos os pontos que pertencem a fronteira do quadrado Q_i com coordenadas inteiras são os pontos do espiral de nivel i. Nós podemos dividir esses pontos em 4 subconjuntos: os pon-

tos do Leste L_i, os pontos do norte N_i, os pontos do oeste O_i e os pontos do sul S_i. Assim nós podemos definir $L_i = \{(i, j), com\ \ j \in \{-i, i\}\}$, os ponto do norte $N_i = \{(j, i), com\ \ j \in \{-i, i\}\}$, os pontos do oeste $O_i = \{(-i, j) com\ \ j \in \{-i, i\}\}$, os pontos do sul $S_i = \{(j, -i), com\ \ j \in \{-i, i\}\}$.

Uma primeira observação é que os vértices dos quadrados pertencem a dois desses quatro subconjuntos. Considere a seguinte definição:

Definição 3.0.1 *O conjunto dos pontos espiral do nível $i \in \mathbb{N}$ será definido por:*

$$E_i = L_i \cup N_i \cup O_i \cup S_i \tag{3.1}$$

Proposição 3.0.1 *A quantidade de pontos espiral no nivel $i \in \mathbb{N}$ é $8i$*

Demonstração 1: Nós temos a seguinte igualdade $\mid L_i \mid = \mid N_i \mid = \mid O_i \mid = \mid S_i \mid = 2i + 1$, onde $\mid o \mid$ significa cardinalidade. Assim nós temos somando os quatro conjuntos $4(2i + 1) = 8i + 4$ pontos, como os vértices são somados duas vezes temos um total de $8i + 4 - 4 = 8i$ pontos.

Agora vamos descrever a dinâmica. Nós começamos no nível zero (composto apenas pelo ponto $(0, 0)$) e caso a simulação seja do modelo M_2, sorteamos uma direção L, N, O, S, caso seja o modelo M_1, seguimos adiante para o próximo passo. Em seguida sorteamos, se temos fluido vermelho, o que corresponde a preencher dois elos em sequência na direção sorteada com a cor vermelha, ou se temos fluido azul, o que corresponde a um elo azul na direção sorteada ou se não temos fluido. Na sequência começamos a percorrer o espiral nos niveis i´s. Sempre se começa no ponto $(i, -i + 1)$, em seguida percorre o espiral no nível i no sentido anti-horário, o ponto final desse nível é o ponto $(i, -i)$. A cada ponto que se percorre no nível i faz-se necessário realizar um teste para saber se existe alguma direção viável para a realização do fluido vermelho ou azul. Não ter direção viável significa que os quatro elos que partem desse ponto contém fluído. Caso seja viável alguma direção, então os sorteios são realizados sucessivamente até tomar uma direção viável com seu respectivo fluido ou não. Caso não exista essa viabilidade, nós passamos para o ponto seguinte do percurso.

Nesse segundo momento, nós vamos estudar os pontos ao norte no nível i, denotando eles por $N_k = (k, i)$ com $k \in -i + 1, i - 1$, vamos supor $k > 0$, mas isso não muda a demonstração.

O ponto-oesteN_k é o próximo ponto na lista e o ponto-lesteN_k é o ponto anterior na lista. Vamos tratar do ponto-sulN_k, fazendo o gráfico precisamos somar cinco parcelas de pontos para sabermos quantos pontos antes de N_k na lista corresponde ao ponto-sulN_k. A primeira parcela é a quantidade de pontos entre N_k e (i, i), ou seja, $i - k$ pontos. Em seguida entre (i, i) e $(i, -i + 1)$ o que corresponde a

$i-(-i+1)=2i-1$, a terceira parcela entre $(-i,i+1)$ e $(-i+1,-i+1)$, um total de 2i - 1 pontos também. A quarta parcela entre $(-i+1,-i+1)$ e $(-i+1,i-1)$, mais uma vez $2i-1$ pontos, a última parcela entre $(-i+1,i-1)$ e $(k,i-1)$ que nos fornece mais $k+i-1$ pontos. Somando as cinco parcelas obtemos um total de $8i-5$ pontos atrás na lista. Por fim, o ponto-norteN_k corresponde a um ponto $8(i+1)-5$ a frente na lista, a demonstração usa a demonstração de ponto-sul considerando que N_k é o ponto-sul$(k,i+1)$.

Assim, sabemos também todos os pontos vizinhos dos pontos ao norte.

Na sequência, nós vamos tratar dos pontos a oeste: $O_k=(-i,k)$ com $k\in[-i+1,i-1](k>)0$. O ponto-sulO_k é o próximo ponto na lista, o ponto-norteO_k é o ponto anterior na lista. Agora nos debrucemos para o ponto-lesteO_k.

Novamente, nós temos cinco parcelas de quantidade de pontos a serem somadas. A primeira entre O_k e o ponto $(-i,i)$ um total de $i-k$ pontos, na sequência entre $(-i,i)$ e (i,i) mais $2i$ pontos. Na terceira parcela, temos os pontos entre (i,i) e $(i,-i+1)$ ou seja, $2i-1$ pontos, na quarta parcela temos os pontos entre $(i,-i+1)$ e $(-i+1,-i+1)$, outros $2i-1$ pontos. Por fim, os pontos entre $(-i+1,-i+1)$ e $(-i+1,k)$, mais $k+i-1$ pontos, somando essas cinco parcelas temos que o ponto-lesteO_k está atrás de O_k $8i-3$ posições. Pelo mesmo raciocinio das outras demonstrações o ponto-oesteO_k está $8(i+1)-3=8i+5$ posições a frente de O_k na lista.

A próxima etapa é referente aos pontos ao sul no nível i. Vamos apelidar esses pontos por $S_k=(k,-i)$ com $k\in[-i+1,i-1]$. O ponto-oesteS_k é o ponto anterior na lista, o ponto-lesteS_k é o próximo ponto na lista. A demonstração para os pontos sul e norte de S_k segue os mesmos passos anteriores, considerando cinco parcelas de pontos. A primeira entre S_k e $(-i,-i)$, um total de $k+i$ pontos, na sequência entre $(-i,-i)$ e $(-i,i)$, mais $2i$ pontos. Depois, entre $(-i,i)$ e (i,i), outros $2i$ pontos. A quarta parcela entre (i,i) e $(i,-i+1)$ contando mais $2i-1$ pontos. A quinta parcela corresponde aos pontos entre $(i,-i+1)$ e $(k,-i+1)$, um total de $i-k$ pontos. Adicionando essas cinco parcelas concluimos que o ponto-norteS_k está $8i-1$ posições anteriores a posição de S_k. Para finalizar o ponto-sulS_k está $8(i+1)-1=8i+7$ posições na frente de S_k na lista (mesma idéia de olhar para o ponto-sulS_k e buscar a distância ao seu ponto-norte (que corresponde ao S_k)

3.0.3 Algoritmo da matriz dos vértices

Vamos começar agora a tratar dos algoritmos para solução do problema sobre percolação. Para isto primeiro vamos definir uma matriz que devera guardar as informações dos pontos onde serão sorteados: direção, fluido, cluster. Primeiro passo é definir a estrutura vértice:

abcissa
ordenada
azul
vermelho
dleste
dnorte
doeste
dsul

A variável abcissa possui o valor do respectivo vértice, assim como a variável ordenada. A variável vermelho só assume dois possíveis valores 0 e 1, se assume o valor 1 indica que existe um caminho de elos vermelhos(*cluster*) da origem até esse vértice, se assumir 0, não teremos tal caminho. O mesmo raciocínio para a variável azul. As variáveis dleste, dnorte, doeste e dsul, que aqui estamos denotando começando com a letra d, para indicar por exemplo que dleste significa direção leste, podem assumir apenas três valores 0, 1, 2. Se assumir o valor 0 é porque não tem nenhum fluido nessa aresta, se assumir o valor 1 temos o fluido azul, e se for 2 temos o fluido vermelho.

Uma função importante é o teste de existência de possível sorteio num determinado sitio para um fluído azul ou fluído vermelho, que é um dos resultados apresentados no próximo capitulo desse trabalho.

3.0.4 Algoritmo da etapa nível 0

Começaremos agora apresentando como será feito no nível 0, lembrando que nesse nível como se trata do primeiro, não é preciso fazer o teste de existência de possível vizinho, pois todas as arestas em todas as direções estarão livres. Nesse nível só temos o vértice com abcissa e ordenada iguais a 0. Faz-se um sorteio de direção L, O, S, N cada direção com igual probabilidade. Em seguida faz-se um sorteio com possíveis valores 0, 1, 2, onde 1 tem chance p de ser sorteado, 2 tem chance q, e o 0 tem $1-p-q$ de chance. Se tivermos qualquer direção sorteada e na sequência temos o valor 0 sorteado, então finaliza nesse momento o nível 0. Se tivermos uma direção sorteada por exemplo a N e o número sorteado for o 1, então a variável dnorte com coordenadas $(0,0)$ assume o valor 1, assim como a variável dsul do vértice com coordenadas $(0,1)$ também recebe o valor 1. Estes mesmos dois vértices tem as variáveis azul atualizadas para o valor 1. Este ultimo procedimento é similar se tivermos outras direções e sorteio igual a 1. A última possíbilidade ocorre quando se sorteia uma direção qualquer, por exemplo N, e o valor sorteado for igual a 2. Nessa situação, as variáveis dnorte do vértice $(0,0)$, dsul do vértice $(0,1)$, dnorte do

vértice $(0,1)$ e dsul do vértice $(0,2)$ recebem o valor 2. As variáveis vermelho dos vértices $(0,0)$, $(0,1)$ e $(0,2)$ recebem o valor 2. Faremos agora o algoritmo o qual chamamos de SIMULA, no ponto $(0,0)$.

SIMULA ponto $P_0 = (0,0)$

Sorteia direção d entre L, N, S, O
Sorteia fluido f entre $A(azul), V(vermelho), B(branco)$ Caso

```
1 d = L e f = A:
2 verticeList[0].dleste = 1
3 verticeList[0].azul = 1
4 verticeList[1].doeste = 1
5 verticeList[1].azul = 1
```

```
1 d = L e f = V:
2 verticeList[0].dleste = 2
3 verticeList[0].vermelho = 1
4 verticeList[1].doeste = 2
5 verticeList[1].dleste = 2
6 verticeList[1].vermelho = 1
7 verticeList[10].doeste = 2
8 verticeList[10].vermelho = 1
```

```
1 d = N e f = A:
2 verticelist[0].dnorte = 1
3 verticelist[0].azul = 1
4 verticelist[3].dsul = 1
5 verticelist[3].azul = 1
```

```
1 d = N e f = V:
2 verticeList[0].dnorte = 2
3 verticeList[0].vermelho = 1
4 verticeList[3].dsul = 2
5 verticeList[3].dnorte = 2
6 verticeList[3].vermelho = 1
7 verticeList[14].dsul = 2
8 verticeList[14].vermelho = 1
```

```
1 d = O e f = A:
2 verticeList[0].doeste = 1
3 verticeList[0].azul = 1
4 verticeList[5].dleste = 1
5 verticeList[5].azul = 1
```

```
1 d = O e f = V:
2 verticeList[0].doeste = 2
3 verticeList[0].vermelho = 1
4 verticeList[5].dleste = 2
5 verticeList[5].doeste = 2
6 verticeList[5].vermelho = 1
7 verticeList[18].dleste = 2
8 verticeList[18].vermelho = 1
```

```
1 d = S e f = A:
2 verticeList[0].dsul = 1
3 verticeList[0].azul = 1
4 verticeList[7].dnorte = 1
5 verticeList[7].azul = 1
```

```
1 d = S e f = V:
2 verticeList[0].dsul = 2
3 verticeList[0].vermelho = 1
4 verticeList[7].dnorte = 2
5 verticeList[7].dsul = 2
6 verticeList[7].vermelho = 1
7 verticeList[22].dnorte = 2
8 verticeList[22].vermelho = 1
```

..

Capítulo 4

Resultados

Apresentaremos a seguir os principais resultados obtidos em nosso estudo.

4.0.1 Viabilidade do modelo 2

Começaremos com o resultado que estabelece a viabilidade para o modelo M_2

Proposição 4.0.1 *Considere $p \in E_i$ um ponto que estamos fazendo o teste para viabilidade de um possível sorteio de direção e de sorteio de fluido. Basicamente, nós temos quatro situações:*

i) Se $p = (i, -i + 1)$ sempre tem direção viável, a direção Sul.

ii) Se $p = (i, k)$ com $k = -i + 2, ..., i - 1$, nós não teremos direção viável a partir de p se e somente se tivermos elos entre $(i - 1, k)$ e $(i + 1, k)$ e entre $(i, k - 1)$ e $(i, k + 1)$ e nessa situação temos fluidos vermelhos.

iii) Se $p = (i, i)$ sempre temos direção viável a direção leste (L),

iv) Pela simetria do problema, nós estabelecemos resultados similares para os pontos em N_i, O_i e S_i, com exceção do último vértice do quadrado Q_i, $p = (i, -i)$, o qual apresenta situação similar a do ponto ii).

Demonstração 2

i) Nesse caso, p é o primeiro ponto do nível i a se fazer o teste de direção viável. O ponto acima de p, o ponto $q = (i, -i + 2)$, ainda não foi testado, assim, os elos entre $(i, -i + 2)$ até $(i, -i)$ não tem fluido, obtendo assim uma direção viável.

ii) Claramente, se os quatro elos que partem de p já estão ocupados por fluidos vermelhos, então não existe direção partindo de p. Por outro lado, se não temos direção viável os quatro elos que partem de p tem algum fluido. Vamos mostrarque tem que ser do tipo vermelho. Se existisse um elo azul partindo de p para a esquerda, ou para cima seria um absurdo, pois isso ocorreria se já tivessemos feito um sorteio de direção no ponto p, e se os elos que partem dos pontos $(i - 1, k)$ ou $(i, k - 1)$

fossem azuis, os elos entre (i,k) e $(i,k+1)$ ou (i,k) e $(i+1,k)$ seriam sem fluido o que resultaria na existência de direções viáveis.

iii) Na situação $p=(i,i)$ o ponto $q=(i+1,i)$ que pertence ao nível $i+1$ não foi ainda percorrido e não pode ter sido alcançado por qualquer outro ponto já investigado. Os únicos pontos que podem alcançar q de nível inferior a $i+1$ é p ou $(i-1,i)$ pontos que ainda não foram percorridos. Assim sempre existe direção viável para esse p

iv) A não existência de direção viável corresponde a existência de fluidos viáveis entre $(i,-i+1)$ até $(i,-i-1)$ e de $(i-1,i)$ até $(i+1,i)$

Abaixo, nós determinamos as coordenadas dos pontos vizinhos a um ponto do espiral e construimos o algoritmo que simula uma realização dos caminhos dos fluidos

4.0.2 Determinação de um ponto vizinho na Lista

Inicialmente nós vamos começar com os pontos a leste no nível i, tais pontos tem como coordenadas $L_k=(i,k)$ onde $k\in[-i+1,i-1]$, estamos descartando aqueles pontos do nível i que são os vértices do quadrado do nível i. Basicamente nós teremos dois casos:

$k=-i+1$ as posições na lista dos quatro vizinhos de L_k são: o ponto-oesteL_k é o ponto anterior na lista, o ponto-norteL_k é o ponto a frente na lista, o ponto-sulL_ké o ponto $8i-1$ unidades a frente e o ponto-lesteL_k é o ponto $8i+1$ a frente. A demonstração é simples basta fazer o desenho e usar o fato de que no nível i temos $8i$ pontos.

$k\in[-i+2,i-1]$ nesse caso vamos supor que $k>0$, mas a demonstração vale para $k\geq 0$ também. O ponto-sulL_k é o ponto anterior na lista, o ponto-norteL_k é o ponto a frente na lista, para o cálculo do ponto-oesteL_k temos de somar a quantidade de pontos entre L_k e seu ponto a oeste. Essa soma é dividida em 5 parcelas: pontos abaixo de L_k são todos os pontos do nível i, onde o último é o ponto $(i,-i+1$ a quantidade de pontos entre eles é $k-i+1$, a segunda parcela está entre os pontos $(-i+1,-i+1)$ e $(i,-i+1)$, total de $2i-1$. A terceira parcela entre os pontos $(-i+1,-i+1)$ e $(-i+1,i-1)$, total de $2i-2$. Na quarta etapa temos os pontos entre $(-i+1,i-1)$ e $(i-1,i-1)$, perfazendo $2i-2$, e a ultima parcela entre os pontos $(i-1,i-1)$ e $(i-1,k)$ que são $i-1-k$. Adicionando essas cinco parcelas obtemos um total de $8i-7$ pontos anteriores a L_k.

A obtenção na lista do ponto-lesteL_k pode ser calculado usando um truque. O ponto a leste de L_k é um ponto de coordenadas $(i+1,k)$ e o ponto a oeste dele é o próprio L_k, mas o cálculo de pontos a oeste de um ponto no leste, já sabemos. O ponto-leste $(i+1,k)$ está a $8(i+1)-7$ posições atrás, isso implica que o ponto-oesteL_k está $8(i+1)-7$ pontos a frente na lista.

Nesse segundo momento, nós vamos estudar os pontos ao norte no nível i, denotando eles por $N_k = (k, i)$ com $k \in -i+1, i-1$, vamos supor $k > 0$, mas isso não muda a demonstração.

O ponto-oesteN_k é o próximo ponto na lista e o ponto-lesteN_k é o ponto anterior na lista. Vamos tratar do ponto-sulN_k, fazendo o gráfico precisamos somar cinco parcelas de pontos para sabermos quantos pontos antes de N_k na lista corresponde ao ponto-sulN_k. A primeira parcela é a quantidade de pontos entre N_k e (i, i), ou seja, $i - k$ pontos. Em seguida entre (i, i) e $(i, -i + 1)$ o que corresponde a $i - (-i + 1) = 2i - 1$, a terceira parcela entre $(-i, i + 1)$ e $(-i + 1, -i + 1)$, um total de 2i - 1 pontos também. A quarta parcela entre $(-i + 1, -i + 1)$ e $(-i + 1, i - 1)$, mais uma vez $2i - 1$ pontos, a última parcela entre $(-i+1, i-1)$ e $(k, i-1)$ que nos fornece mais $k+i-1$ pontos. Somando as cinco parcelas obtemos um total de $8i-5$ pontos atrás na lista. Por fim, o ponto-norteN_k corresponde a um ponto $8(i+1)-5$ a frente na lista, a demonstração usa a demonstração de ponto-sul considerando que N_k é o ponto-sul$(k, i + 1)$.

Assim, sabemos também todos os pontos vizinhos dos pontos ao norte.

Na sequência, nós vamos tratar dos pontos a oeste: $O_k = (-i, k)$ com $k \in [-i + 1, i - 1](k >)0$. O ponto-sul$O_k$ é o próximo ponto na lista, o ponto-norteO_k é o ponto anterior na lista. Agora nos debrucemos para o ponto-lesteO_k.

Novamente, nós temos cinco parcelas de quantidade de pontos a serem somadas. A primeira entre O_k e o ponto $(-i, i)$ um total de $i - k$ pontos, na sequência entre $(-i, i)$ e (i, i) mais $2i$ pontos. Na terceira parcela, temos os pontos entre (i, i) e $(i, -i+1)$ ou seja, $2i-1$ pontos, na quarta parcela temos os pontos entre $(i, -i+1)$ e $(-i + 1, -i + 1)$, outros $2i - 1$ pontos. Por fim, os pontos entre $(-i + 1, -i + 1)$ e $(-i + 1, k)$, mais $k + i - 1$ pontos, somando essas cinco parcelas temos que o ponto-lesteO_k está atrás de O_k $8i - 3$ posições. Pelo mesmo raciocinio das outras demonstrações o ponto-oesteO_k está $8(i + 1) - 3 = 8i + 5$ posições a frente de O_k na lista.

A próxima etapa é referente aos pontos ao sul no nível i. Vamos apelidar esses pontos por $S_k = (k, -i)$ com $k \in [-i + 1, i - 1]$. O ponto-oesteS_k é o ponto anterior na lista, o ponto-lesteS_k é o próximo ponto na lista. A demonstração para os pontos sul e norte de S_k segue os mesmos passos anteriores, considerando cinco parcelas de pontos. A primeira entre S_k e $(-i, -i)$, um total de $k + i$ pontos, na sequência entre $(-i, -i)$ e $(-i, i)$, mais $2i$ pontos. Depois, entre $(-i, i)$ e (i, i), outros $2i$ pontos. A quarta parcela entre (i, i) e $(i, -i + 1)$ contando mais $2i - 1$ pontos. A quinta parcela corresponde aos pontos entre $(i, -i+1)$ e $(k, -i+1)$, um total de $i-k$ pontos. Adicionando essas cinco parcelas concluimos que o ponto-norteS_k está $8i-1$ posições anteriores a posição de S_k. Para finalizar o ponto-sulS_k está $8(i + 1) - 1 = 8i + 7$ posições na frente de S_k na lista (mesma idéia de olhar para o ponto-sulS_k e buscar

a distância ao seu ponto-norte (que corresponde ao S_k)

Vamos finalizar com os pontos $NE = (i, i)$, $NO = (-i, i)$, $SO = (-i, -i)$ e $SE = (i, -i)$. Nós temos ponto-oesteNE é o próximo na lista, o ponto-sulNE é o ponto anterior na lista, o ponto-norteNE corresponde a um ponto norte no nível $i+1$, sendo assim o ponto NE corresponde a um ponto sul de algum ponto ao norte do nível $i + 1$. Logo a distância entre eles é $8(i + 1) - 5$.

Por fim, o ponto-lesteNE é um ponto a leste no nível $(i+1)$. Consequentemente, o ponto NE é um ponto a oeste de um outro ponto a leste do nível $i + 1$ e tendo a distância entre eles de $8(i + 1) - 7$. Agora, nós vamos tratar do ponto NO. O ponto-lesteNO é o ponto anterior na lista, o ponto-sulNO é o próximo ponto na lista. O ponto-norteNO é um ponto ao norte do nível $i+1$ que tem NO como ponto sul, sendo assim sua distância é de $8(i + 1) - 5$ e o ponto-oesteNO é um ponto a oeste no nível $i + 1$ que tem NO como ponto a leste, sendo assim a distância entre eles é de $8(i + 1) - 3$. Considere agora, o ponto SO, seu ponto ao norte é o anterior na lista, o seu ponto a leste é o próximo na lista. O ponto-oesteNO é um ponto nível $i+1$ que tem NO como ponto a leste, portanto tem uma distância de $8(i+1)-3$, o ponto a sul de NO é um ponto sul no nível $i+1$ que tem NO como ponto ao norte, o que produz uma distância entre eles de $8(i + 1) - 1$. Olhemos para SO, o ponto a norte de SO é anterior na lista, o ponto a sul de SO é um ponto sul do nível $i + 1$ que tem SO como ponto ao norte definindo uma distância de $8(i + 1) - 1$ entre eles, e o ponto a leste de SO é o próximo ponto na lista. E o ponto a oeste de SO é um ponto a oeste do nível $i + 1$ tendo SO como ponto a leste produzindo uma distância $8i + 5$. Por fim, o ponto SE. O ponto ao norte do ponto SE é o ponto inicial do nível i, ou seja está a uma distância $8i - 1$ na lista dos vértices. O ponto a oeste é o ponto anterior na lista, o ponto a leste é o ponto seguinte na lista e o ponto a sul de SE é um ponto sul do nível $i + 1$ que tem SE como ponto ao norte, ou seja, tem como distância $8i + 7$. Sabemos que no nível i temos $8i$ pontos, assim a quantidade de pontos até o nível i é dado pela soma: $1 + 8(1) + 8(2) + .. + 8(i - 1)$, usando a fórmula de uma progressão aritmética essa soma vale: $1 + 4i(i - 1)$. Então, todos os pontos do nível i podem ser expressos como sendo da forma $1 + 4i(i - 1) + j$, com $j = 0, ..., 8(i - 1)$. Se j = 0 temos o ponto $(i, -i + 1)$, se $j = 1, .., (2i - 2)$ temos os pontos ao leste L_k, se $j = 2i-1$ obtemos o ponto NE. Ao variarmos $j = 2i, ...(4i-2)$ extraimos os pontos ao norte N_k. O ponto NO aparece com $j = 4i - 1$ e os pontos ao oeste, O_k são descritos quando $j = 4i, ..., 6i - 2$. O ponto à sudoeste SO tem $j = 6i - 1$. Se j varia de $6i$ até $8i - 2$, nós temos os pontos ao sul. Por fim, o ponto sudeste SE tem $j = 8i - 1$.

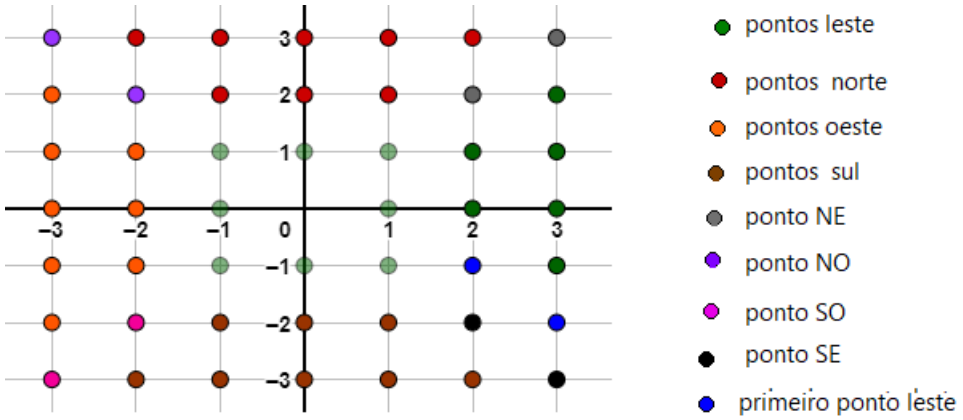


Figura 4.1: pontos marcados a partir do nivel 2

.

4.0.3 Algoritmo Principal e funções

O algoritmo principal está descrito abaixo, o mesmo se utiliza de algumas funçoes que são descritas na sequência. Estas funções são chamadas no algoritmo principal pela expressão SIMULA.

Testa se a variável cluster = 1, para algum elemento da fronteira, para que possamos saber se o cluster tocou a fronteira.

4.0.4 SIMULA PONTO P_i

Coordenada $(i, -i + 1)$, é o primeiro ponto a leste, temos que $P = 1 + 4i(i - 1)$

```
1  ALGORITMO PRINCIPAL
2  DEFINE N número de níveis;
3  SIMULA ponto 0 da lista P_0 = (0, 0);
4  SIMULA pontos no nível 1;
5  para nível i = 2, ..., N faça
6  |  SIMULA ponto 1 + 4i(i − 1) da lista de vértices;
7  |  para j = 1, ..., 2i − 2 faça
8  |  |  SIMULA ponto 1 + 4i(i − 1) + j da lista de vértices.(pontos ao leste)
9  |  fim
10 |  para  j = 2i − 1 faça
11 |  |  SIMULA ponto 1 + 4i(i − 1) + 2i + j (ponto nordeste = NE)
12 |  fim
13 |  para j = 2i, ..., 4i − 2 faça
14 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(pontos ao norte)
15 |  fim
16 |  para j = 4i − 1 faça
17 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(ponto noroeste
   |  |    = NO)
18 |  fim
19 |  para j = 4i, ...., 6i − 2 faça
20 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(pontos a oeste)
21 |  fim
22 |  para j = 6i − 1 faça
23 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(ponto sudoeste
   |  |    = SO)
24 |  fim
25 |  para j = 6i, ..., 8i − 2 faça
26 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(pontos ao sul)
27 |  fim
28 |  para j = 8i − 1 faça
29 |  |  SIMULA ponto 1 + 4i(i − 1) + j na lista de vértices.(ponto suldeste
   |  |    = SE)
30 |  fim
31 |  ATUALIZA CLUSTERS GERADOS ATÉ O NÍVEL i
32 fim
```

```
1  PRIMEIRO PONTO A LESTE
2  d = L e f = A
3  verticeList[1 + 4i(i − 1)].dleste = 1
4  verticeList[1 + 4i(i − 1) + (8i + 1)].doeste = 1
5  se verticeList[1 + 4i(i − 1)].azul = 1 ou
     verticeList[1 + 4i(i − 1) + (8(i + 1)].azul = 1 então
6  |  verticeList[1 + 4i(i − 1)].azul = 1 verticeList[1 + 4i(i − 1) + (8i + 1)].azul
   |    = 1
7  senão
8  d = L e f = V
9  verticeList[1 + 4i(i − 1)].dleste = 2
10 verticeList[1 + 4i(i − 1) + (8i + 1)].doeste = 2
11 verticeList[1 + 4i(i − 1) + (8i + 1)].dleste = 2
12 verticeList[1 + 4i(i − 1) + (8i + 1) + 8(i + 1) + 1].doeste = 2
13 se  verticeList[1 + 4i(i − 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + (8i + 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + (8i + 1) + 8(i + 1) + 1].vermelho = 1 então
14 |  verticeList[1 + 4i(i − 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + (8i + 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + (8i + 1) + 8(i + 1) + 1].vermelho = 1
15 senão
16 d = N e f = A
17 verticeList[1 + 4i(i − 1)].dnorte = 1
18 verticeList[1 + 4i(i − 1) + 1].dsul = 1
19 se verticeList[1 + 4i(i − 1)].azul = 1 ou verticeList[1 + 4i(i − 1) + 1].azul
     = 1 então
20 |  verticeList[1 + 4i(i − 1)].azul = 1 verticeList[1 + 4i(i − 1) + 1].azul = 1
21 senão
22 d = N e f = V
23 verticeList[1 + 4i(i − 1)].dnorte = 2
24 verticeList[1 + 4i(i − 1) + 1].dsul = 2
25 verticeList[1 + 4i(i − 1) + 1].dnorte = 2
26 verticeList[1 + 4i(i − 1) + 2].dsul = 2
27 se verticeList[1 + 4i(i − 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + 1].vermelho = 1 ou ou
     verticeList[1 + 4i(i − 1) + 2].vermelho = 1 então
28 |  verticeList[1 + 4i(i − 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + 1].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + 2].vermelho = 1
29 senão
```

```
1  d = O e f = A
2  verticeList[1 + 4i(i − 1)].doeste = 1
3  verticeList[1 + 4i(i − 1) − 1].dleste = 1
4  se verticeList[1 + 4i(i − 1)].azul = 1 ou verticeList[1 + 4i(i − 1) − 1].azul
     = 1 então
5  |  verticeList[1 + 4i(i − 1)].azul = 1 verticeList[1 + 4i(i − 1) − 1].azul = 1
6  senão
7  d = O e f = V
8  verticeList[1 + 4i(i − 1)].doeste = 2
9  verticeList[1 + 4i(i − 1) − 1].dleste = 2
10 verticeList[1 + 4i(i − 1) − 1].doeste = 2
11 verticeList[1 + 4i(i − 1) − 2].dleste = 2
12 se verticeList[1 + 4i(i − 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) − 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) − 2].vermelho = 1 então
13 |  verticeList[1 + 4i(i − 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) − 1].vermelho = 1
   |    verticeList[1 + 4i(i − 1) − 2].vermelho = 1
14 senão
15 d = S e f = A
16 verticeList[1 + 4i(i − 1)].dsul = 1
17 verticeList[1 + 4i(i − 1) + (8i − 1)].dnorte = 1
18 se verticeList[1 + 4i(i − 1)].azul = 1 ou
     verticeList[1 + 4i(i − 1) + (8i − 1)].azul = 1 então
19 |  verticeList[1 + 4i(i − 1)].azul = 1 verticeList[1 + 4i(i − 1) + (8i − 1)].azul
   |    = 1
20 senão
21 d = S e f = V
22 verticeList[1 + 4i(i − 1)].dsul = 2
23 verticeList[1 + 4i(i − 1) + (8i − 1)].dsul = 2
24 verticeList[1 + 4i(i − 1) + (8i − 1)].dnorte = 2
25 verticeList[1 + 4i(i − 1) + (8i − 1) + (8(i + 1) − 1)].dnorte = 2
26 se verticeList[1 + 4i(i − 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + (8i − 1)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + (8i − 1) + (8(i + 1) − 1)].vermelho = 1 então
27 |  verticeList[1 + 4i(i − 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + (8i − 1)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + (8i − 1) + (8(i + 1) − 1)].vermelho = 1
28 senão
```

Coordenada (i, k), com $k = (-i + 2), ..(i - 1)$ são os pontos a leste, temos que $P = 1 + 4i(i - 1) + j$, com $j = 1, ..., (2i - 2)$

```
1  PONTOS A LESTE
2  d = L e f = A
3  verticeList[1 + 4i(i − 1) + j].dleste = 1
4  verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].doeste = 1
5  se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
   |   verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].azul = 1
7  senão
8  d = L e f = V
9  verticeList[1 + 4i(i − 1) + j].dleste = 2
10 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].doeste = 2
11 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].dleste = 2
12 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7) + (8(i + 2) − 7)].doeste = 2
13 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7) + (8(i + 2) − 7)].vermelho = 1
     então
14 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7) + (8(i + 2) − 7)].vermelho = 1
15 senão
```

```
1  d = N e f = A verticeList[1 + 4i(i − 1) + j].dnorte = 1
2  verticeList[1 + 4i(i − 1) + j + 1].dsul = 1
3  se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
     verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
4  |  verticeList[1 + 4i(i − 1) + j].azul = 1
   |   verticeList[1 + 4i(i − 1) + j + 1].azul = 1
5  senão
6  d = N e f = V
7  para j = 1, ...(2i − 3) e i = 3, ... faça
8  |  ;
9  verticeList[1 + 4i(i − 1) + j].dnorte = 2
10 verticeList[1 + 4i(i − 1) + j + 1].dnorte = 2
11 verticeList[1 + 4i(i − 1) + j + 1].dsul = 2
12 verticeList[1 + 4i(i − 1) + j + 2].dsul = 2
13 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + 2].vermelho = 1 então
14 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + 2].vermelho = 1
15 senão
16 d = N e f = V
17 para j = (2i − 2) faça
18 |  ;
19 verticeList[1 + 4i(i − 1) + j].dnorte = 2
20 verticeList[1 + 4i(i − 1) + j + 1].dnorte = 2
21 verticeList[1 + 4i(i − 1) + j + 1].dsul = 2
22 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 4))].dsul = 2
23 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 4))].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1
   |   verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 4))].vermelho = 1
25 senão
```

```
1  d = O e f = A
2  verticeList[1 + 4i(i − 1) + j].doeste = 1
3  verticeList[1 + 4i(i − 1) + j − (8i − 7)].dleste = 1
4  se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
       verticeList[1 + 4i(i − 1) + j − (8i − 7)].azul = 1 então
5      verticeList[1 + 4i(i − 1) + j].azul = 1
         verticeList[1 + 4i(i − 1) + j − (8i − 7)].azul = 1
6  senão
7      d = O e f = V
8      para j = 1 faça
9          ;
10     verticeList[1 + 4i(i − 1) + j].doeste = 2
11     verticeList[1 + 4i(i − 1) + j − (8i − 7)].dleste = 2
12     verticeList[1 + 4i(i − 1) + j − (8i − 7)].doeste = 2
13     verticeList[1 + 4i(i − 1) + j − (8i − 7) − 1].dleste = 2
14     se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
           verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1 ou
           verticeList[1 + 4i(i − 1) + j − (8i − 7) − 1].vermelho = 1 então
15         verticeList[1 + 4i(i − 1) + j].vermelho = 1
             verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1
             verticeList[1 + 4i(i − 1) + j − (8i − 7) − 1].vermelho = 1
16     senão
17         d = O e f = V
18         para j = 2, ..(2i − 3) e i = 3, ... faça
19             ;
20         verticeList[1 + 4i(i − 1) + j].doeste = 2
21         verticeList[1 + 4i(i − 1) + j − (8i − 7)].dleste = 2
22         verticeList[1 + 4i(i − 1) + j − (8i − 7)].doeste = 2
23         verticeList[1 + 4i(i − 1) + j − (8i − 7) − (8(i − 1) − 7)].dleste = 2
24         se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
               verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1 ou
               verticeList[1 + 4i(i − 1) + j − (8i − 7) − (8(i − 1) − 7)].vermelho = 1 então
25             verticeList[1 + 4i(i − 1) + j].vermelho = 1
                 verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1
                 verticeList[1 + 4i(i − 1) + j − (8i − 7) − (8(i − 1) − 7)].vermelho = 1
26     senão
```

```
1  d = O e f = V
2  para j = (2i − 2) faça
3  |  ;
4  verticeList[1 + 4i(i − 1) + j].doeste = 2
5  verticeList[1 + 4i(i − 1) + j − (8i − 7)].dleste = 2
6  verticeList[1 + 4i(i − 1) + j − (8i − 7)].doeste = 2
7  verticeList[1 + 4i(i − 1) + j − (8i − 7) + 1].dleste = 2
8  se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − (8i − 7) + 1].vermelho = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − (8i − 7)].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − (8i − 7) + 1].vermelho = 1
10 senão
11 d = S e f = A
12 verticeList[1 + 4i(i − 1) + j].dsul = 1
13 verticeList[1 + 4i(i − 1) + j − 1].dnorte = 1
14 Caso verticeList[1 + 4i(i − 1) + j].azul = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1].azul = 1 verticeList[1 + 4i(i − 1) + j].azul
     = 1 verticeList[1 + 4i(i − 1) + j − 1].azul = 1
```

Coordenada (i, i), é o ponto nordeste, temos que $P = 1 + 4i(i - 1 + j)$, com $j = (2i - 1)$

```
1  d = S e f = V
2  para j = 1 faça
3  |  ;
4  verticeList[1 + 4i(i − 1) + j].dsul = 2
5  verticeList[1 + 4i(i − 1) + j − 1].dnorte = 2
6  verticeList[1 + 4i(i − 1) + j − 1].dsul = 2
7  verticeList[1 + 4i(i − 1) + j − 1 + (8i − 1)].dnorte = 2
8  se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1 + (8i − 1)].vermelho = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − 1 + (8i − 1)].vermelho = 1
10 senão
11 d = S e f = V
12 para j = 2, ..(2i − 2) faça
13 |  ;
14 verticeList[1 + 4i(i − 1) + j].dsul = 2
15 verticeList[1 + 4i(i − 1) + j − 1].dnorte = 2
16 verticeList[1 + 4i(i − 1) + j − 1].dsul = 2
17 verticeList[1 + 4i(i − 1) + j − 2].dnorte = 2
18 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 2].vermelho = 1 então
19 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j − 2].vermelho = 1
20 senão
```

```
1  PONTO NORDESTE
2  d = L e f = A
3  verticeList[1 + 4i(i − 1) + j].dleste = 1
4  verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].doeste = 1
5  se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
   |    verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].azul = 1
7  senão
8  d = L e f = V
9  verticeList[1 + 4i(i − 1) + j].dleste = 2
10 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].doeste = 2
11 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].dleste = 2
12 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)) + (8(i + 2) − 7))].doeste = 2
13 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)) + (8(i + 2) − 7))].vermelho = 1
     então
14 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7))].vermelho = 1
   |    verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 7)) + (8(i + 2) − 7))].vermelho
   |    = 1
15 senão
16 d = N e f = A
17 verticeList[1 + 4i(i − 1) + j].dnorte = 1
18 verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].dsul = 1
19 se verticeList[1 + 4i(i − 1) + j].azul= 1 ou
     verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].azul = 1 então
20 |  verticeList[1 + 4i(i − 1) + j].azul = 1
   |    verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].azul = 1
21 senão
```

```
1  d = N e f = V
2  verticeList[1 + 4i(i − 1) + j].dnorte = 2
3  verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].dsul = 2
4  verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].dnorte = 2
5  verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5) + (8(i + 2) − 5))].dsul = 2
6  se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
       verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].vermelho = 1 ou
       verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5) + (8(i + 2) − 5))].vermelho = 1
       então
7      verticeList[1 + 4i(i − 1) + j].vermelho = 1
         verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5)].vermelho = 1
         verticeList[1 + 4i(i − 1) + j + (8(i + 1) − 5) + (8(i + 2) − 5))].vermelho
         = 1
8  senão
9      d = O e f = A
10     verticeList[1 + 4i(i − 1) + j].doeste = 1
11     verticeList[1 + 4i(i − 1) + j + 1].dleste = 1
12     se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
           verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
13         verticeList[1 + 4i(i − 1) + j].azul = 1
             verticeList[1 + 4i(i − 1) + j + 1].azul = 1
14     senão
15         d = O e f = V
16         verticeList[1 + 4i(i − 1) + j].doeste = 2
17         verticeList[1 + 4i(i − 1) + j + 1].dleste = 2
18         verticeList[1 + 4i(i − 1) + j + 1].doeste = 2
19         verticeList[1 + 4i(i − 1) + j + 2].dleste = 2
20         se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
               verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 ou
               verticeList[1 + 4i(i − 1) + j + 2].vermelho então
21             verticeList[1 + 4i(i − 1) + j].vermelho = 1
                 verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1
                 verticeList[1 + 4i(i − 1) + j + 2].vermelho = 1
22         senão
```

```
1  d = S e f = A
2  verticeList[1 + 4i(i − 1) + j].dsul = 1
3  verticeList[1 + 4i(i − 1) + j − 1].dnorte = 1
4  se verticeList[1 + 4i(i − 1) + j].azul = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
5  |  verticeList[1 + 4i(i − 1) + j].azul = 1
   |   verticeList[1 + 4i(i − 1) + j − 1].azul = 1
6  senão
7  d = S e f = V
8  verticeList[1 + 4i(i − 1) + j].dsul = 2
9  verticeList[1 + 4i(i − 1) + j − 1].dnorte = 2
10 verticeList[1 + 4i(i − 1) + j − 1].dsul = 2
11 verticeList[1 + 4i(i − 1) + j − 2].dnorte = 2
12 se verticeList[1 + 4i(i − 1) + j].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 ou
     verticeList[1 + 4i(i − 1) + j − 2].vermelho = 1 então
13 |  verticeList[1 + 4i(i - 1) + j].vermelho = 1 verticeList[1 + 4i(i - 1) + j -
   |   1].vermelho = 1 verticeList[1 + 4i(i - 1) + j - 2].vermelho = 1
14 senão
```

por simetria os casos dos pontos do Norte, Noroeste, oeste, Suldoeste e Suldeste seguem o mesmo padrão

Uma função importante é o teste de existência de possível sorteio num determinado sitio para um fluído azul ou fluído vermelho

Teste Possíveis Vizinhos

Para fazer esse teste num ponto P qualquer precisamos verificar se $verticeList[P].dleste = 0$ ou $verticeList[P].doeste = 0$ ou $verticeList[P].dnorte = 0$ ou $verticeList[P].dsul = 0$. Se pelo menos uma dessas condições é verdadeira, o teste éverdadeiro. Sendo assim, nós podemos ficar fazendo sorteios, de direção e fluído, até que num determinado momento, nós tenhamos um sorteio válido.

Sorteios Válidos

Supondo que o testePossívelVizinho para um vértice P seja verdadeiro, ou seja, existe sorteios válidos. Nós ficamos fazendo sorteios até obtermos um sorteio válido. Abaixo segue o pseudocódigo para termos um sorteio válido:

```
1  TestePossivelVizinho(P) = verdade
2  SorteioValido = falso
3  enquanto SorteioValido = falso faça
4  |  Sorteio direção = d, sorteio fluido = f
5  fim
6  SorteioValido = falso:
7  para P = (i, −i + 1),se d = L e f = A e verticeList[P].dleste = 0 então
8  |  sorteioValido = verdade
9  senão
10 fim
11  faça
12 fim
13 se d = L e f = V e verticeList[P].dleste = 0 então
14 |  sorteiovalido = verdade
15 senão
16 fim
17 se d = N e f = A e verticeList[P].dnorte = 0 então
18 |  sorteiovalido = verdade
19 senão
20 fim
21 se d = N e f = V e verticeList[P].dnorte = 0 então
22 |  sorteiovalido = verdade
23 senão
24 fim
25 se d = O e f = A e verticeList[P].doeste = 0 então
26 |  sorteiovalido = verdade
27 senão
28 fim
29 se d = O e f = V e verticeList[P].doeste = 0 e verticeList[P − 1].doeste = 0
    então
30 |  sorteiovalido = verdade
31 senão
32 fim
33 se d = S e f = A então
34 |  sorteiovalido = verdade
35 senão
36 fim
37 se d = S e f = V então
38 |  sorteiovalido = verdade
39 senão
40 fim
```

Pseudo-código para encontrar sorteio valido

```
1  para P é um ponto a leste do nível i diferente de (i, −i + 1), se d = L e
     f = A e verticeList[P].dleste = 0 então
2  |  sorteioValido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste= 0 e verticeList[P + 8i + 1].dleste
     = 0 então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 e verticeList[P + 1].dnorte
     = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste = 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e
     verticeList[P − 8i + 7].doeste = 0 então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P − 1].dsul = 0 e
     então
24 |  sorteiovalido = verdade
25 senão
```

```
1  para P é o ponto NE = (i, i), se d = L e f = A e verticeList[P].dleste = 0
     então
2  |  sorteiovalido = verdade
3  senão
4  |  faça
5  |  se d = L e f = V e verticeList[P].dleste = 0 então
6  |  |  sorteiovalido = verdade
7  |  senão
8  |  |  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  |  |  sorteiovalido = verdade
10 |  |  senão
11 |  |  |  se d = N e f = V e verticeList[P].dnorte = 0 e
   |  |  |     verticeList[P + 8i + 3].dnorte = 0 então
12 |  |  |  |  sorteiovalido = verdade
13 |  |  |  senão
14 |  |  |  |  se d = O e f = A então
15 |  |  |  |  |  sorteiovalido = verdade
16 |  |  |  |  senão
17 |  |  |  |  |  se d = O e f = V então
18 |  |  |  |  |  |  sorteiovalido = verdade
19 |  |  |  |  |  senão
20 |  |  |  |  |  |  se d = S e f = A e verticeList[P].dsul = 0 então
21 |  |  |  |  |  |  |  sorteiovalido = verdade
22 |  |  |  |  |  |  senão
23 |  |  |  |  |  |  |  se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P − 1].dsul = 0
   |  |  |  |  |  |  |     então
24 |  |  |  |  |  |  |  |  sorteiovalido = verdade
25 |  |  |  |  |  |  |  senão
```

```
1  para P é um ponto a norte do nível i,se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteioValido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste = 0 e verticeList[P − 1].dleste= 0
     então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste = 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e verticeList[P + 1].doeste = 0
     então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P − 8i + 5].dsul
     = 0 então
24 |  sorteiovalido = verdade
25 senão
```

```
1  para Case P é o ponto NO = (−i, i), se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteiovalido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste = 0 e verticeList[P − 1].dleste = 0
     então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 e
     verticeList[P + 8i + 3].dnorte = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste = 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e
     verticeList[P + 8i + 5].doeste = 0 então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P + 1].dsul = 0
     então
24 |  sorteiovalido = verdade
25 senão
```

```
1  para P é um ponto a oeste do nível i, se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteioValido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste = 0 e verticeList[P − 8i + 3].dleste
     = 0 então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 e verticeList[P − 1].dnorte
     = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste= 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e
     verticeList[P + 8i + 5].doeste = 0 então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P + 1].dsul = 0
     então
24 |  sorteiovalido = verdade
25 senão
```

```
1  para Case P é o ponto SO = (−i, −i), se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteiovalido = verdade
3  senão
4  |  faça
5  |  se d = L e f = V e verticeList[P].dleste = 0 e verticeList[P + 1].dleste = 0
   |  então
6  |  |  sorteiovalido = verdade
7  |  senão
8  |  |  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  |  |  sorteiovalido = verdade
10 |  |  senão
11 |  |  |  se d = N e f = V e verticeList[P].dnorte = 0 e verticeList[P − 1].dnorte
   |  |  |  = 0 então
12 |  |  |  |  sorteiovalido = verdade
13 |  |  |  senão
14 |  |  |  |  se d = O e f = A e verticeList[P].doeste = 0 então
15 |  |  |  |  |  sorteiovalido = verdade
16 |  |  |  |  senão
17 |  |  |  |  |  se d = O e f = V e verticeList[P].doeste = 0 e
   |  |  |  |  |  verticeList[P + 8i + 5].doeste = 0 então
18 |  |  |  |  |  |  sorteiovalido = verdade
19 |  |  |  |  |  senão
20 |  |  |  |  |  |  se d = S e f = A e verticeList[P].dsul = 0 então
21 |  |  |  |  |  |  |  sorteiovalido = verdade
22 |  |  |  |  |  |  senão
23 |  |  |  |  |  |  |  se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P + 8i + 7].dsul = 0
   |  |  |  |  |  |  |  então
24 |  |  |  |  |  |  |  |  sorteiovalido = verdade
25 |  |  |  |  |  |  |  senão
```

```
1  para P é um ponto a sul do nível i, se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteioValido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste = 0 então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 e
     verticeList[P + 8i − 1].dnorte = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste= 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e verticeList[P − 1].doeste = 0
     então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 e verticeList[P + 8i + 7].dsul = 0
     então
24 |  sorteiovalido = verdade
25 senão
```

```
1  para Case P é o ponto SE = (i, −i), se d = L e f = A e
     verticeList[P].dleste = 0 então
2  |  sorteiovalido = verdade
3  senão
4   faça
5  se d = L e f = V e verticeList[P].dleste = 0 então
6  |  sorteiovalido = verdade
7  senão
8  se d = N e f = A e verticeList[P].dnorte = 0 então
9  |  sorteiovalido = verdade
10 senão
11 se d = N e f = V e verticeList[P].dnorte = 0 e
     verticeList[P − 8i + 1].dnorte = 0 então
12 |  sorteiovalido = verdade
13 senão
14 se d = O e f = A e verticeList[P].doeste = 0 então
15 |  sorteiovalido = verdade
16 senão
17 se d = O e f = V e verticeList[P].doeste = 0 e verticeList[P − 1].doeste = 0
     então
18 |  sorteiovalido = verdade
19 senão
20 se d = S e f = A e verticeList[P].dsul = 0 então
21 |  sorteiovalido = verdade
22 senão
23 se d = S e f = V e verticeList[P].dsul = 0 então
24 |  sorteiovalido = verdade
25 senão
```

4.0.5 Atualiza Clusters do nível i

Ao simular os pontos do nível i, pode se ter uma alteração nas variáveis de cluster azul e vermelho para pontos dos níveis $i-2$ e $i-1$ e do próprio nível i. Sendo assim, faz-se necessário atualizar essas variáveis de cores dos clusters dos pontos desses níveis. Abaixo descrevemos como se faz isso:

PARA $j = 1, ..., (8i-1)$ Testar se o ponto $1 + 4i(i-1) + j$ do nível i tem algum vizinho com elo vermelho entre eles e se um desses vizinhos pertence ao cluster

vermelho. Se isso for verdade, a variável cluster vermelho desse ponto deve valer 1.

O pseudo código para ATUALIZAR O NÍVEL i fica:

```
1  para j = 1, 2i − 2 faça
2  |  Execute
3  fim
4  ;
5  se verticeList[1 + 4i(4i − 1) + j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + j].dnorte = 1 então
6  |  verticeList[1 + 4i(4i − 1) + j + 1].azul = 1]
7  senão
8  fim
9  se verticeList[1 + 4i(4i − 1) + j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + j].dnorte = 2 então
10 |  verticeList[1 + 4i(4i − 1) + j + 1].vermelho = 1]
11 senão
12 fim
```

```
1  para j = (2i − 1), ...(4i − 2) faça
2  |  Execute
3  ;
4  se verticeList[1 + 4i(4i − 1) + j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + j].doeste = 1 então
5  |  verticeList[1 + 4i(4i − 1) + j + 1].azul = 1]
6  senão
7  se verticeList[1 + 4i(4i − 1) + j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + j].doeste = 2 então
8  |  verticeList[1 + 4i(4i − 1) + j + 1].vermelho = 1]
9  senão
```

```
1  para j = (4i − 1), ...(6i − 2) faça
2  |  Execute
3  ;
4  se verticeList[1 + 4i(4i − 1) + j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + j].dsul = 1 então
5  |  verticeList[1 + 4i(4i − 1) + j + 1].azul = 1]
6  senão
7  se verticeList[1 + 4i(4i − 1) + j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + j].dsul = 2 então
8  |  verticeList[1 + 4i(4i − 1) + j + 1].vermelho = 1]
9  senão
```

```
1  para j = (6i − 1), ...(8i − 2) faça
2  |  Execute
3  ;
4  se verticeList[1 + 4i(4i − 1) + j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + j].dleste = 1 então
5  |  verticeList[1 + 4i(4i − 1) + j + 1].azul = 1]
6  senão
7  se verticeList[1 + 4i(4i − 1) + j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + j].dleste = 2 então
8  |  verticeList[1 + 4i(4i − 1) + j + 1].vermelho = 1]
9  senão
10 se verticeList[1 + 4i(4i − 1) + (8i − 1)].azul = 1 e
     verticeList[1 + 4i(4i − 1) + (8i − 1)].dnorte = 1 então
11 |  verticeList[1 + 4i(4i − 1) + 1].azul = 1]
12 senão
13 se verticeList[1 + 4i(4i − 1) + (8i − 1)].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + (8i − 1)].dnorte = 2 então
14 |  verticeList[1 + 4i(4i − 1) + 1].vermelho = 1]
15 senão
```

Agora vejamos o código no sentido inverso

```
1  para j = 1, 2i − 2 faça
2  |  Execute
3  fim
4  ;
5  se verticeList[1 + 4i(4i − 1) + 8i − j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + 8i − j].doeste = 1 então
6  |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].azul = 1]
7  senão
8  fim
9  se verticeList[1 + 4i(4i − 1) + 8i − j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + 8i − j].doeste = 2 então
10 |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].vermelho = 1]
11 senão
12 fim
```

```
1  para j = (2i − 1), ...(4i − 2) faça
2  |  Execute
3  ;
4  se verticeList[1 + 4i(4i − 1) + 8i − j].azul = 1 e
     verticeList[1 + 4i(4i − 1) + 8i − j].dnorte = 1 então
5  |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].azul = 1]
6  senão
7  se verticeList[1 + 4i(4i − 1) + 8i − j].vermelho = 1 e
     verticeList[1 + 4i(4i − 1) + 8i − j].dnorte = 2 então
8  |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].vermelho = 1]
9  senão
```

```
1 para j = (4i − 1), ..., (6i − 2) faça
2 |  Execute
3 ;
4 se verticeList[1 + 4i(4i − 1) + 8i − j].azul = 1 e
    verticeList[1 + 4i(4i − 1) + 8i − j].dleste = 1 então
5 |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].azul = 1]
6 senão
7 se verticeList[1 + 4i(4i − 1) + 8i − j].vermelho = 1 e
    verticeList[1 + 4i(4i − 1) + 8i − j].dleste = 2 então
8 |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].vermelho = 1]
9 senão
```

```
1 para j = (6i − 1), ...(8i − 2) faça
2 |  Execute
3 ;
4 se verticeList[1 + 4i(4i − 1)8i − j].azul = 1 e
    verticeList[1 + 4i(4i − 1) + 8i − j].dleste = 1 então
5 |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].azul = 1]
6 senão
7 se verticeList[1 + 4i(4i − 1) + 8i − j].vermelho = 1 e
    verticeList[1 + 4i(4i − 1) + 8i − j].dsul = 2 então
8 |  verticeList[1 + 4i(4i − 1) + 8i − j − 1].vermelho = 1]
9 senão
10 se verticeList[1 + 4i(4i − 1) + 1].azul = 1 e
    verticeList[1 + 4i(4i − 1) + 1].dsul = 1 então
11 |  verticeList[1 + 4i(4i − 1) + 8i].azul = 1]
12 senão
13 se verticeList[1 + 4i(4i − 1) + 8i].vermelho = 1 e
    verticeList[1 + 4i(4i − 1) + 8i].doeste = 2 então
14 |  verticeList[1 + 4i(4i − 1) + 1].vermelho = 1]
15 senão
```

Também deve ser feito o teste de se existe um elo azul entre esse ponto e algum de seus vizinhos e se um desses pontos pertence ao cluster azul, caso isto ocorra a variável cluster azul do ponto em questão deve valer 1.

FIM PARA

Para realizar o teste acima, têm-se que fazer o teste com os pontos a leste do

nível i, depois o ponto NE, os pontos ao norte, o ponto NO, os pontos a oeste, o ponto SO, os pontos a sul e o ponto SE. Nós temos que fazer a separação nos casos acima, pois os vizinhos nas listas de vértices tem regras diferentes dependendo dessas situações.

Nível 0

Neste nível só temos um ponto, o ponto 0 na lista de vértices que corresponde ao ponto com coordenadas $(0, 0)$. A atualização do cluster é feito da seguinte forma.

```
1  se verticeList[0].dleste = 1 e verticeList[1].azul = 1 então
2  |  verticeList[0.azul = 1]
3  senão
4  fim
5  se verticeList[0].dnorte = 1 e verticeList[3].azul = 1 então
6  |  verticeList[0.azul = 1]
7  senão
8  fim
9  se verticeList[0].doeste = 1 e verticeList[5].azul = 1 então
10 |  verticeList[0.azul = 1]
11 senão
12 fim
13 se verticeList[0].dsul = 1 e verticeList[7].azul = 1 então
14 |  verticeList[0.azul = 1]
15 senão
16 fim
17 se verticeList[0].dleste = 2 e verticeList[1].vermelho = 1 então
18 |  verticeList[0.vermelho = 1]
19 senão
20 fim
21 se verticeList[0].dnorte = 2 e verticeList[3].vermelho = 1 então
22 |  verticeList[0.vermelho = 1]
23 senão
24 fim
25 se verticeList[0].doeste = 2 e verticeList[5].vermelho = 1 então
26 |  verticeList[0.vermelho = 1]
27 senão
28 fim
29 se verticeList[0].dsul = 2 e verticeList[7].vermelho = 1 então
30 |  verticeList[0.vermelho = 1]
31 senão
32 fim
```

Nível i maior que 2

Nesse caso os pontos do nível i, são expressos da forma padrão $Pj = 1+4i(i-1)+j$ com $j = 0, ..., (8i-1)$

Começamos com $j = 0$

```
1  para j = 1, ..., (2i − 2) ( agora é a etapa dos pontos a leste faça
2  se verticeList[1 + 4i(i − 1)].dleste = 1 e
     verticeList[1 + 4i(i − 1) + 8i + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1)].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1)].dnorte = 1 e verticeList[1 + 4i(i − 1) + 1].azul
     = 1 então
6  |  verticeList[1 + 4i(i − 1)].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1)].doeste = 1 e
     verticeList[1 + 4i(i − 1) − 8i + 7].azul = 1 então
9  |  verticeList[1 + 4i(i − 1)].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1)].dsul = 1 e verticeList[1 + 4i(i − 1) + 8i].azul = 1
     então
12 |  verticeList[1 + 4i(i − 1)].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1)].dleste = 2 e
     verticeList[1 + 4i(i − 1) + 8i + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1)].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1)].dnorte = 2 e
     verticeList[1 + 4i(i − 1) + 1].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1)].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1)].doeste = 2 e
     verticeList[1 + 4i(i − 1) − 8i + 7].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1)].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1)].dsul = 2 e
     verticeList[1 + 4i(i − 1) + 8i − 1].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = 1, ..., (2i − 2) (agora é a etapa dos pontos a leste faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j − 8i + 7].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i − 1].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j − 8i + 7].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = (2i − 1) ponto NE faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
25 senão
```

```
1  para j = 2i, ..., (4i − 2) (agora é a etapa dos pontos ao norte faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j − 8i + 5].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j − 8i + 5].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = (4i − 1) (agora é a etapa do ponto NO faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
     verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
     verticeList[1 + 4i(i − 1) + j + 8i + 3].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
     verticeList[1 + 4i(i − 1) + j + 8i + 5].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
     verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
     verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
     verticeList[1 + 4i(i − 1) + j + 8i + 3].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
     verticeList[1 + 4i(i − 1) + j + 8i + 5].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
     verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = 4i, ..., (6i − 2) (agora é a etapa dos pontos a oeste faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j − 8i + 5].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 3].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j − 8i + 5].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = 6i − 1 (agora é a etapa do ponto SO faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
     verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
     verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
     verticeList[1 + 4i(i − 1) + j + 8i + 5].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
     verticeList[1 + 4i(i − 1) + j + 8i + 7].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
     verticeList[1 + 4i(i − 1) + j + 6i + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
     verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
     verticeList[1 + 4i(i − 1) + j + 8i + 5].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
     verticeList[1 + 4i(i − 1) + j + 8i + 7].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = 6i, ..., (8i − 2) (agora é a etapa dos pontos ao sul faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j − 8i + 1].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 7].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) + j − 8i + 1].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 7].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

```
1  para j = (8i − 1) (agora é a etapa do ponto SE faça
2  se verticeList[1 + 4i(i − 1) + j].dleste = 1 e
       verticeList[1 + 4i(i − 1) + j + 1].azul = 1 então
3  |  verticeList[1 + 4i(i − 1) + j].azul = 1
4  senão
5  se verticeList[1 + 4i(i − 1) + j].dnorte = 1 e
       verticeList[1 + 4i(i − 1) + j − 8i + 1].azul = 1 então
6  |  verticeList[1 + 4i(i − 1) + j].azul = 1
7  senão
8  se verticeList[1 + 4i(i − 1) + j].doeste = 1 e
       verticeList[1 + 4i(i − 1) + j − 1].azul = 1 então
9  |  verticeList[1 + 4i(i − 1) + j].azul = 1
10 senão
11 se verticeList[1 + 4i(i − 1) + j].dsul = 1 e
       verticeList[1 + 4i(i − 1) + j + 8i + 7].azul = 1 então
12 |  verticeList[1 + 4i(i − 1) + j].azul = 1
13 senão
14 se verticeList[1 + 4i(i − 1) + j].dleste = 2 e
       verticeList[1 + 4i(i − 1) + j + 1].vermelho = 1 então
15 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
16 senão
17 se verticeList[1 + 4i(i − 1) + j].dnorte = 2 e
       verticeList[1 + 4i(i − 1) − j + 1].vermelho = 1 então
18 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
19 senão
20 se verticeList[1 + 4i(i − 1) + j].doeste = 2 e
       verticeList[1 + 4i(i − 1) + j − 1].vermelho = 1 então
21 |  verticeList[1 + 4i(i − 1) + j].vermelho = 1
22 senão
23 se verticeList[1 + 4i(i − 1) + j].dsul = 2 e
       verticeList[1 + 4i(i − 1) + j + 8i + 7].vermelho = 1 então
24 |  verticeList[1 + 4i(i − 1)].vermelho = 1
25 senão
```

Nível 1

O nível $i = 1$ pode ser feito pelo caso acima, considerando que não temos os pontos a leste. Ou seja, se o nível i é igual a 1, coloca-se um desvio na etapa dos pontos ao leste, não entrando nesse laço de repetição.

4.0.6 ATUALIZA CLUSTERS ATÉ O NÍVEL i

Um fato a se observar é que após diversas simulações de níveis poderemos ter mais de um cluster de qualquer uma das cores que não tocam a origem, e poderemos ter somente um de cada cor que toca também a origem. Ao simular o nível i pode-se ligar mais de um cluster ao cluster que toca a origem. Esse fenômeno pode provocar a necessidade de atualizar os vértices até o nível 1. O algoritmo ATUALIZA CLUSTERS ATÉ O NÍVEL i, como o nome diz atualiza todos os vértices que não pertenciam ao cluster da origem ao cluster da origem, onde essa ligação fica estabelecida na simulação do nível i. Nesta subseção destacamos uma parte de potencial expansão e aprofundamento em relação ao escopo do algoritmo proposto. A função ATUALIZA CLUSTERS ATÉ O NÍVEL i, embora tenha sido planejada para ser integrada como parte vital do processo, não foi abordada neste trabalho devido as limitações de tempo e recursos. No entanto essa lacuna não apenas apresenta oportunidades empolgantes para investigações futuras, mas também pode impactar positivamente os resultados e a eficácia geral do algoritmo proposto.

Capítulo 5

Conclusão

Em suma, a presente dissertação explorou avanços na simulação de um modelo de percolação de elos abertos com dois fluidos e interdependência espacial. Ao longo deste estudo foi desenvolvido uma estrutura de dados, que armazena as informações dos sorteios que aconteceram nas coodernadas de um espiral, e começamos a desenvolver a função que atualiza os clusters e avalia se houve percolação. O que permite uma análise detalhada das propriedades emergentes desse sistema complexo.

Os resultados obtidos não apenas contribuem para uma compreensão mais profunda do modelo adotado, mas também abrem caminho para aplicações práticas que podem ser nos mais diversos campos como da engenharia, fisica, geologia e estudo de fluxo em meios porosos.

No entanto, é importante ressaltar que, embora este trabalho tenha alcançado resultados promissores, ainda há espaço para aprimoramentos e extensões futuras. A incorporação de fatores adicionais, como o uso de redes neurais para investigação da probabilidade de percolação, ou a consideração de outras geometrias como por exemplo as redes hexagonais

Em última análise, esta dissertação ressalta a importância contínua da pesquisa em modelos de percolação e sua relevância para a compreensão de fenômenos complexos em sistemas naturais e tecnológicos. Ao fornecer uma ferramenta poderosa para explorar a dinâmica de sistemas de percolação de elos abertos e dependência espacial, este estudo oferece uma contribuição significativa para o avanço da ciência e abre novos horizontes para investigações futuras nessa emocionante área de pesquisa

Referências Bibliográficas

[1] BROADBENT, S. R., HAMMERSLEY, J. M., 1957, "Percolation processes: I. Crystals and mazes". In: *Mathematical proceedings of the Cambridge philosophical society*, v. 53, pp. 629–641. Cambridge University Press.

[2] FERREIRA, R., 2022, *Deep Learning*. 1 ed. São Paulo, Conteudo Saraiva.

[3] FONTES, L., 1996, "Percolação, notas do IMPA", *Rio de Janeiro*.

[4] GAO, J., BULDYREV, S. V., STANLEY, H. E., et al., 2012, "Networks formed from interdependent networks", *Nature physics*, v. 8, n. 1, pp. 40–48.

[5] MENDES, G. A., 2011, "Estudo de sistemas complexos com interações de longo alcance: percolação, redes e tráfego", .

[6] OLIVEIRA, M. M. D., BRAGA, G. A., 2002, "O Fenômeno de Transição de Fase no Modelo de Percolação de Elos* em d Dimensões", *Revista Brasileira de Ensino de Física*, v. 24, pp. 448–454.

[7] ROSS, S., 2009, *Probabilidade: um curso moderno com aplicações*. Bookman Editora.

[8] TANAKA, N. I., TAVARES, H. R., 1993, "Introdução a teoria da percolação", .

[9] YATES, R. D., GOODMAN, D. J., 2014, *Probability and stochastic processes: a friendly introduction for electrical and computer engineers*. John Wiley & Sons.