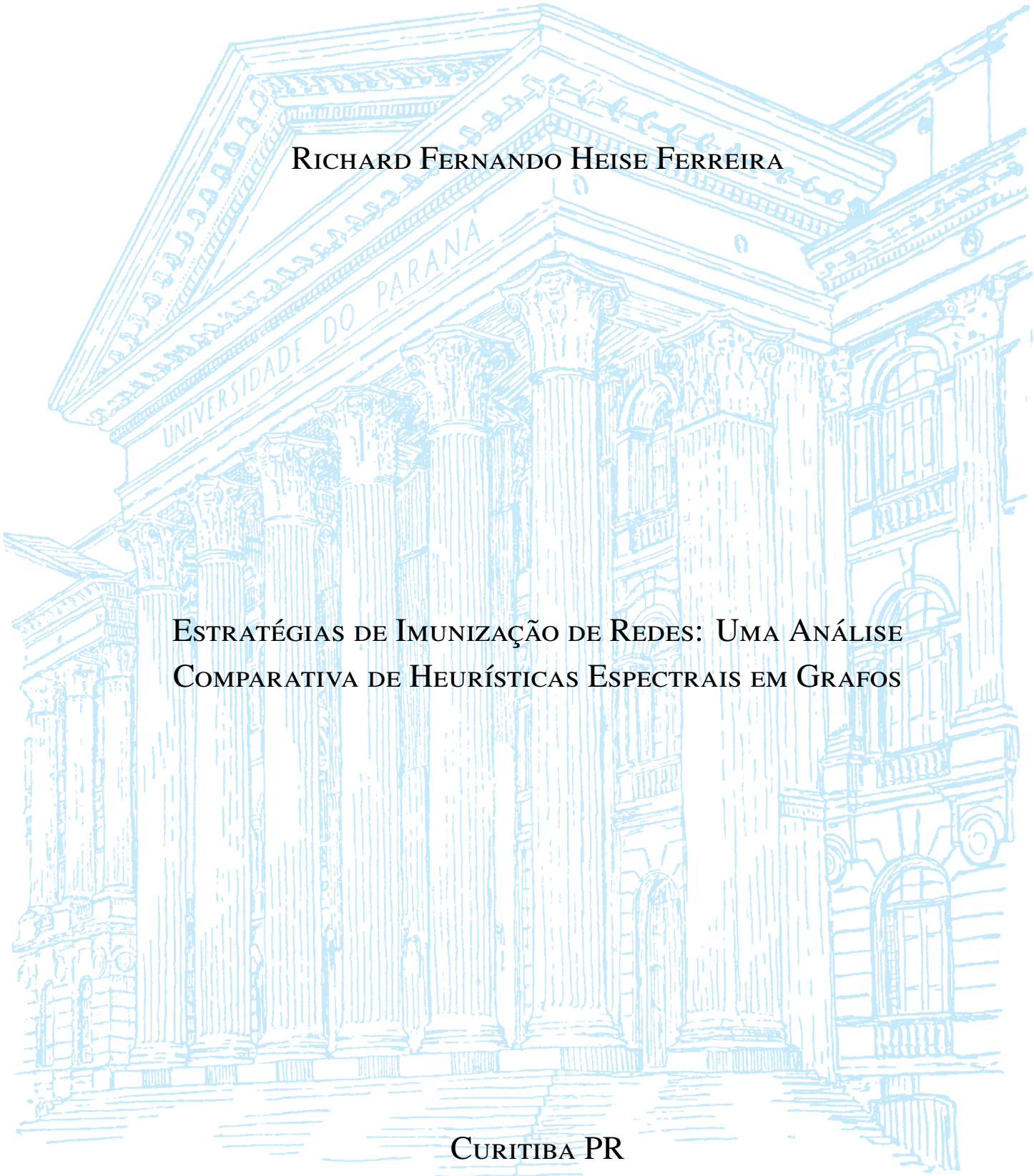
Universidade Federal do Paraná

Richard Fernando Heise Ferreira



Estratégias de Imunização de Redes: Uma Análise Comparativa de Heurísticas Espectrais em Grafos

Curitiba PR
2024

Richard Fernando Heise Ferreira

Estratégias de Imunização de Redes: Uma Análise Comparativa de Heurísticas Espectrais em Grafos

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação pela Universidade Federal do Paraná, setor de Ciências Exatas.

Área de concentração: *Ciência da Computação*.

Orientador: André Luiz Pires Guedes.

Curitiba PR
2024

Ao meu avô, meu nono, Bonifácio Raymundo Nadolny, que me ensinou a amar e agora reside em meus sonhos mais tranquilos e felizes.

Agradecimentos

Aos meus inesquecíveis e indispensáveis amigos.
Aos meus pais, por terem me dado condição de estudo, com seu esforços inacabáveis.
Ao C3SL, pela formação científica e profissional, além do auxílio necessário para a estadia na Universidade.
À música, minha companheira inseparável.

"Computador... Computador... Computador, é tudo computador! (...)"
Julinho da Van, Choque de Cultura.

"O Brasil ainda não tem tecnologia para dar um soco sério."
Renan, Choque de Cultura.

"Enquanto eu corria
Assim eu ia
Lhe chamar
Enquanto corria a barca
Lhe chamar"
Os Novos Baianos, Preta Pretinha

"Capitalism has triumphed all over the world, but this triumph is only the prelude to the triumph of labour over capital."
Lenin, The Three Sources and Three Component Parts of Marxism

Resumo

Utilizando a Teoria de Grafos, é possível modelar interações complexas entre pessoas ou dispositivos, gerando objetos matemáticos que representam diversos tipos de conexões ou a ausência delas. Em outras palavras, podemos utilizar a matemática para definir e analisar redes, capturando a essência de como diferentes entidades estão interconectadas. Essa abordagem revela-se particularmente útil quando consideramos a propagação de vírus, sejam eles biológicos, como doenças contagiosas, ou artificiais, como vírus de computador. Esses agentes se aproveitam da natureza interconectada do nosso mundo para se espalharem eficientemente e atingirem o maior número possível de entidades em uma rede. Contudo, a mesma estrutura de redes que permite a disseminação de ameaças também possibilita a criação de estratégias de defesa eficazes. A imunização, seja ela biológica ou digital, visa proteger as redes ao reduzir sua vulnerabilidade a ataques. No entanto, um desafio surge: como podemos imunizar uma rede de forma mais eficaz possível, considerando um número limitado de recursos, como vacinas? Para responder a essa questão, recorremos à Álgebra Linear em conjunto com a Teoria de Grafos, culminando na Teoria Espectral de Grafos. Essa abordagem oferece ferramentas matemáticas para medir a vulnerabilidade de uma rede, especialmente por meio do estudo de autovalores e autovetores da matriz de adjacência do grafo correspondente. Ao longo da última década, diversos algoritmos foram desenvolvidos para abordar esse problema, utilizando estratégias aproximadas que se baseiam nas ferramentas da Teoria Espectral. É fundamental, portanto, classificar e comparar essas estratégias em termos de eficácia na imunização e eficiência no tempo de execução. Além disso, surge a questão de quais dessas estratégias podem ser expandidas ou aprimoradas em estudos futuros. Neste trabalho, apresentamos uma análise de algumas dessas estratégias, incluindo uma modificação em um dos algoritmos que demonstrou resultados promissores em comparação com a heurística original. Concluímos que, embora alguns algoritmos se destaquem em determinados cenários, existe um campo vasto para futuras melhorias, tanto dentro quanto fora do domínio da Álgebra Linear.

Palavras-chave: imunização de redes; grafos; teoria espectral dos grafos; algoritmos de aproximação; álgebra linear.

Abstract

Using Graph Theory, it is possible to model complex interactions between people or devices, generating mathematical objects that represent different types of connections or their absence. In other words, we can use mathematics to define and analyze networks, capturing the essence of how different entities are interconnected. This approach proves particularly useful when considering the spread of viruses, whether biological, such as contagious diseases, or artificial, such as computer viruses. These agents take advantage of the interconnected nature of our world to spread efficiently and reach as many entities as possible in a network. However, a network structure that allows the same transmission of threats also allows the creation of effective defense strategies. Immunization, whether biological or digital, aims to protect networks by reducing their vulnerability to attacks. However, a challenge arises: how can we immunize a network as effectively as possible, given a limited number of resources such as vaccines? To answer this question, we resort to Linear Algebra in conjunction with Graph Theory, culminating in Spectral Graph Theory. This approach offers mathematical tools to measure the vulnerability of a network, especially through the study of eigenvalues and eigenvectors of the adjacency matrix of the corresponding graph. Over the last decade, several algorithms have been developed to address this problem, using approximate strategies that are based on the tools of Spectral Theory. It is essential, therefore, to classify and compare these strategies in terms of effectiveness in immunization and efficiency in execution time. Furthermore, the question arises as to which of these strategies can be expanded or improved in future studies. In this work, we present an analysis of some of these strategies, including a modification to one of the algorithms that demonstrated promising results compared to the original heuristic. We conclude that, although some algorithms stand out in certain scenarios, there is a vast field for future improvements, both inside and outside the domain of Linear Algebra.

Keywords: network immunization; graphs; spectral graph theory; approximation algorithms; linear algebra.

Lista de Figuras

1.1 Grafo de uma rede social simples. Os nodos representam pessoas e as arestas suas amizades. 14
1.2 Interações entre os personagens de *Les Miserábles*. 14

2.1 Grafo estrela. 23

3.1 Ciclo 4 29
3.2 Ciclo 4 sem o vértice C 29

5.1 Representação do grafo netshield. Em cinza estão os nodos imunizados pelo algoritmo 1. Adaptado de Chen et al. (2016) 38
5.2 Representação do grafo netshield imunizado pelo Walk-4 e Walk4-aprimorado. 39
5.3 Representação do grafo netshield imunizado pelo X-NB. 39
5.4 Gráfico de autoqueda por número de recursos para o grafo gen_graph, 50 vértices e 100 arestas. 40
5.5 Gráfico de recursos por tempo para o grafo gen_graph. 50 vértices e 100 arestas. Pode ser lido como “quanto tempo cada algoritmo levou para produzir uma autoqueda”. 41
5.6 Gráfico de autoqueda por número de recursos para o grafo árvore gen_tree, 50 vértices e 100 arestas. 41
5.7 Gráfico de recursos por tempo para o grafo árvore gen_tree. 50 vértices e 100 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda". 42
5.8 Gráfico de autoqueda por número de recursos para o grafo netshield. 16 vértices e 15 arestas. 42
5.9 Gráfico de recursos por tempo para o grafo netshield. 16 vértices e 15 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda". 43
5.10 Gráfico de autoqueda por número de recursos para o grafo seguidores. 2536 vértices e 3921 arestas. 43
5.11 Gráfico de recursos por tempo para o seguidores. 2536 vértices e 3921 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda". 44
5.12 Gráfico de autoqueda por número de recursos para o grafo web-spam. 4767 vértices e 37375 arestas. 44
5.13 Gráfico de recursos por tempo para o web-spam. 4767 vértices e 37375 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda". 45

5.14 Gráfico de autoqueda por número de recursos para o grafo tech-WHOIS. 7476 vértices e 56943 arestas. 45
5.15 Gráfico de recursos por tempo para o tech-WHOIS. 4767 vértices e 37375 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda". 46

Lista de Tabelas

1.1 Matriz de adjacência da figura 1.1 15
1.2 Exemplo de medida de 'vulnerabilidade' em grafos. Mais arestas tornam o grafo mais bem conectado e, portanto, mais vulnerável. Note que a cadeia (a) e a estrela (b) têm o mesmo número de arestas, mas a medida λ_1 considera corretamente a estrela como mais vulnerável. Adaptado de Chen et al. (2016) 16

4.1 Características do BruteForce 32
4.2 Características do NetShield 33
4.3 Características do NetShield+ 33
4.4 Características do Walk-4 34
4.5 Características do Walk-4 34
4.6 Características do X-NB 35
4.7 Características dos diferentes algoritmos 35

5.1 Grafos utilizados para os testes. 37

Lista de Acrônimos

Eigenvalue	Autovalor
Eigenvector	Autovetor
Eigendrop	Autoqueda
Autoqueda	Diferença entre o maior autovalor do grafo original e o maior autovalor do grafo final

Lista de Símbolos

G	Um grafo qualquer i.e. uma rede
$A(G)$	Uma matriz de adjacência do grafo G, também utilizada como simplesmente A
n	Número de vértices de G
m	Número de arestas de G
r	Taxa de recuperação de um nodo
f	Taxa de infecção de um vírus
k	Quantidade de recursos disponíveis
τ	Limiar epidemiológico
$\lambda_1(A(G))$	Primeiro (o maior) autovalor da matriz de adjacência do grafo G
λ_1	Simplificação do símbolo $\lambda_1(A(G))$
b	Variável do passo do NetShield+
S	Conjunto de nodos a serem imunizados

Sumário

1 Introdução **13**
- 1.1 Grafos . . . 13
- 1.2 Teoria Espectral dos Grafos . . . 15
- 1.3 Epidemiologia . . . 15
- 1.4 O problema . . . 17

2 Revisão da Literatura **19**
- 2.1 NetShield e NetShield+ . . . 19
- 2.2 Walk-4 . . . 22
- 2.3 Non-backtracking X-Centrality (X-NB) . . . 26

3 Walk-4 Aprimorado **29**

4 Implementações **32**
- 4.1 BruteForce . . . 32
- 4.2 NetShield e NetShield+ . . . 33
- 4.3 Walk-4 . . . 34
- 4.4 Walk-4 Aprimorado . . . 34
- 4.5 X-NB . . . 34
- 4.6 Resumindo . . . 35

5 Testes **36**
- 5.1 Método . . . 36
- 5.2 Gráficos . . . 37
- 5.3 Resultados . . . 40

6 Conclusão **47**
- 6.1 Discussão . . . 47
- 6.2 Trabalhos futuros . . . 47

Referências Bibliográficas **49**

Capítulo 1

Introdução

Todos os dias, nos envolvemos em uma infinidade de interações humanas em diferentes contextos. No ambiente de trabalho, na universidade, na escola, no bairro ou mesmo online, com pessoas do mundo todo. Somos seres sociais (Boyd e Richerson, 2009) e, portanto, só conseguimos desenvolver os avanços, tecnologias e descobertas que temos hoje graças às nossas relações sociais. Contudo, nossa sociabilidade pode ser explorada como uma vulnerabilidade.

Do ponto de vista de um vírus, quanto mais interagimos uns com os outros, maior a chance de perpetuação desse vírus – seja ele natural, como uma doença, ou artificial, como um vírus de computador ou propaganda viral. Por conseguinte é natural que um esforço seja feito para conter esse tipo de ameaça: na área da saúde temos vacinas que fortalecem nosso sistema imunológico, prevenindo infecções contagiosas; na computação, contamos com softwares antivírus que detectam e neutralizam ameaças digitais.

No entanto, ainda enfrentamos desafios significativos. Por exemplo, dado um número de vacinas limitadas, pode ser difícil determinar quem deve ser vacinado em uma população, mesmo ignorando questões etárias e pessoas com necessidades especiais, para evitar uma epidemia. Ou quais computadores devem ser monitorados e atualizados para evitar que uma rede inteira seja infectada. Essas questões se tornam ainda mais complexas quando consideramos grandes comunidades ou redes globais. Como podemos modelar esses problemas e identificar as entidades-chave para imunizarmos a fim de diminuir, ou até evitar, a propagação de um vírus em uma rede?

1.1 Grafos

Para entender melhor o problema, é possível abordá-lo pelas lentes da Teoria de Grafos. Um grafo é um objeto matemático que busca modelar relações quaisquer, ou a falta delas. Em sua forma gráfica pode ser interpretado como um desenho em que cada entidade é representada por uma bola, chamada de vértice ou nodo, e as conexões entre esses vértices são linhas, chamadas de arestas. Portanto, é fácil visualizar uma rede, como a figura 1.1, que simboliza uma rede simples de amizade entre pessoas. Como não há peso nas arestas, isto é, todas as amizades valem o mesmo, chamamos esse grafo de simples em vez de ponderado. Além disso, todas as amizades são bidirecionais, ou seja, se o vértice A é amigo de B, B também é amigo de A, não havendo qualquer noção de ordem ou ranqueamento, dessa forma, chamamos o grafo de não-direcionado.

Também fica perceptível como grafos podem escalar com facilidade, uma vez que para criar uma relação basta adicionar uma aresta entre dois vértices, e para criar uma entidade basta adicionar um novo vértice e suas relações. Contudo, apesar a visualização gráfica de um grafo ser interessante para alguns campos específicos, como o estudo de grafos planares (que

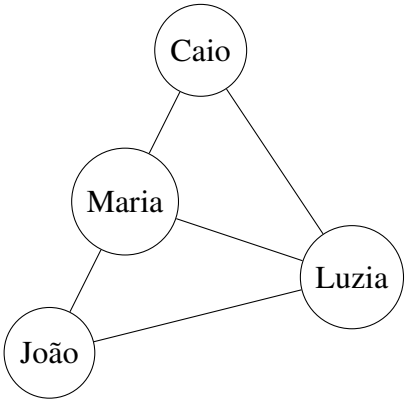


Figura 1.1: Grafo de uma rede social simples. Os nodos representam pessoas e as arestas suas amizades.

podem ser desenhados em um plano sem intersecção de arestas), o desenho pode ser confuso ou simplesmente ineficiente para nos ajudar a entender as relações de uma rede complexa ou suficientemente grande. O grafo de interações entre os personagens do livro *Les Misérables*, de Victor Hugo, exemplificado na figura 1.2 (Rhodes e Johnson, 2021), nos dá a noção de um grafo grande em que seu desenho não necessariamente contribui para o entendimento detalhado do sistema, apesar dos esforços do artista em tornar algumas informações, como quantidade de vizinhos, dada pelo tamanho do nodo, mais acentuada no desenho.

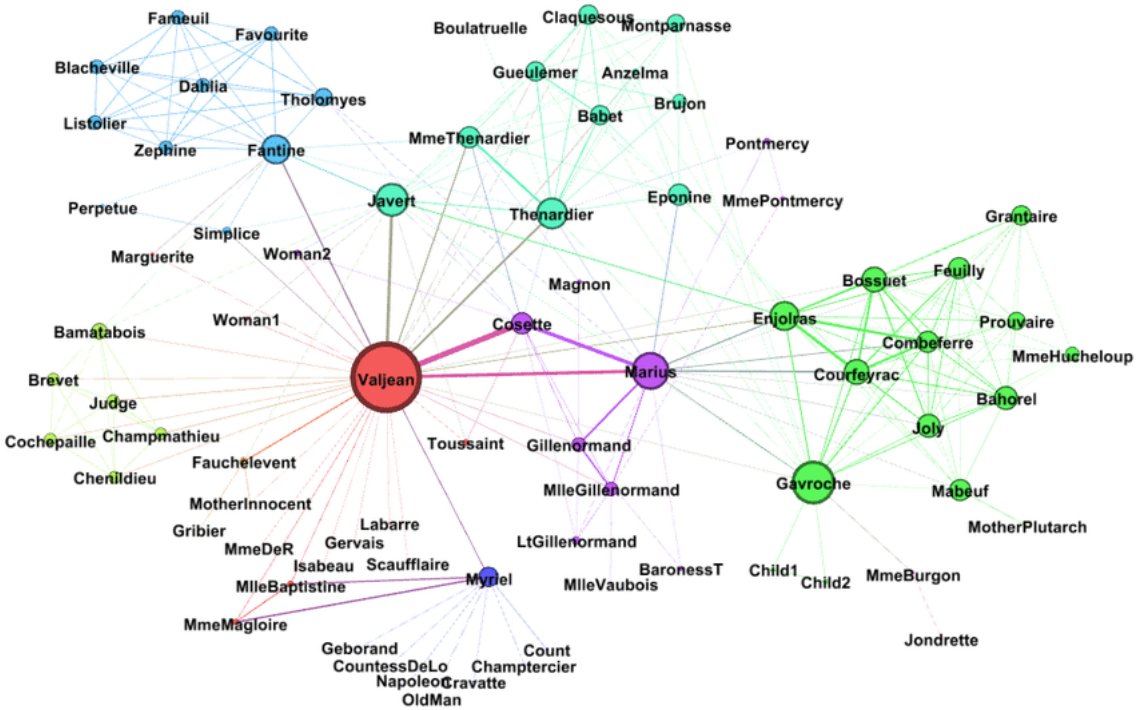


Figura 1.2: Interações entre os personagens de *Les Misérables*.

Com isso em mente, é importante que sejamos capazes de desenvolver outra forma de modelar relações, outra forma de ver um grafo. Uma dessas formas de descrever essas relações entre os vértices é através de uma matriz de adjacência, por exemplo, em que cada linha e cada coluna representa um vértice do grafo. Quando os vértices são vizinhos, i.e. existe uma aresta

entre eles, colocamos o valor 1 na célula de intersecção dos vértices e o valor 0 nas demais, como exemplificado na tabela 1.1.

Tabela 1.1: Matriz de adjacência da figura 1.1

Nodos	Maria	João	Caio	Luzia
Maria	0	1	1	1
João	1	0	0	1
Caio	1	0	0	1
Luzia	1	1	1	0

A representação matricial oferece uma maneira poderosa de descrever grafos complexos, destacando suas relações intrínsecas. Além disso, sua utilidade vai além da mera visualização: as matrizes permitem a aplicação de operações matemáticas que revelam informações importantes sobre o grafo. Ao unir os conceitos de grafos, matrizes e Álgebra Linear, surge a Teoria Espectral dos Grafos. Essa interseção de campos busca aprofundar nossa compreensão dos grafos ao analisar matrizes derivadas de suas estruturas. Explorando os autovalores e autovetores associados às diversas matrizes, não somente de adjacência, possíveis geradas a partir de um grafo qualquer. Essa abordagem oferece ferramentas fundamentais na métrica de vulnerabilidade de uma rede, como abordamos a seguir.

1.2 Teoria Espectral dos Grafos

A álgebra linear possui poderosas ferramentas para executar operações em vetores. Vetores são, computacionalmente, uma sequência de números[1]. Uma matriz pode ser entendida como um vetor de vetores, isto é, um vetor de sequências de números. No estudo dos vetores e matrizes é possível realizar uma operação linear em um vetor, ou seja, aplicar uma função T com uma determinada lei de formação que recebe de entrada um vetor $\vec{v}$ arbitrário e terá como saída um vetor $\vec{v'}$ (é importante ressaltar que, nesse caso, a função T terá sempre a mesma dimensão de domínio e contra-domínio, pois usaremos sempre matrizes quadradas). Dessa forma, caso haja uma proporcionalidade entre $\vec{v}$ e $\vec{v'}$, o valor λ que descreve a proporcionalidade é chamado de autovalor e $\vec{v}$ passa a ser o autovetor associado à λ, ou seja, se $T(\vec{v}) = \lambda\vec{v} = \vec{v'}$ é satisfeito, λ é um autovalor e $\vec{v}$ é um autovetor associado à λ.

O conceito de "espectralidade" propõe a utilização de autovalores e autovetores para descobrir informações importantes sobre a matriz que está sendo estudada. A Teoria Espectral dos Grafos é o ato de aplicar essa ideia nas matrizes possíveis de um grafo. Comumente (Butler, 2022) as matrizes mais utilizadas para esse estudo são a Laplaciana e a Matriz de Adjacência. As aplicações são inúmeras (Spielman, 2007), no entanto, é usada principalmente a matriz de adjacência neste trabalho.

1.3 Epidemiologia

A Teoria dos Grafos é relativamente nova – creditada ao trabalho das pontes de Königsberg por Euler em 1736 – em comparação com a álgebra linear, que remonta a cerca

[1]Essa definição é suficiente para essa aplicação. Apesar disso, o campo da álgebra linear, propriamente dito, define vetor de outras formas, como uma classe de equipolência de segmentos de reta orientados, que possuem todos a mesma intensidade (também designada por norma ou módulo), mesma direção e mesmo sentido.

de 1650 a.C. Essas duas frentes da matemática se encontram por volta de 1960, como Teoria Espectral dos Grafos. Algo parecido acontece com o estudo formalizado da epidemiologia, que começa por volta do século XIX, e com a Teoria Espectral dos Grafos, muito mais recente. Os epidemiologistas empregam uma série de conceitos para investigar a propagação de infecções em redes, sendo os modelos cascata arbitrários mais reconhecidos denominados SIS (Suscetível-Infectado-Suscetível) e SIR (Suscetível-Infectado-Recuperado) (Keeling e Eames, 2005). Ambos os modelos compartilham algumas taxas fundamentais aplicáveis a qualquer rede estudada em epidemiologia: cada vírus apresenta uma taxa de infecção f, cada nodo possui uma taxa de recuperação r, e toda rede tem um limiar epidemiológico τ. Além disso, há considerações adicionais, como taxas de natalidade e mortalidade, que estão além do escopo deste trabalho. Um elemento implícito é o fator temporal, onde a taxa de infecção viral pode ser definida, por exemplo, pelo número de nodos vizinhos infectados durante uma unidade arbitrária de tempo. No modelo SIS, toda entidade de uma rede, i.e. todo vértice de um grafo, possui somente dois estados: suscetível ou infectado. Já no modelo SIR existe um terceiro estado, recuperado, que garante que, enquanto um vértice estiver nesse estado, ele não poderá ser infectado novamente até que volte a se tornar suscetível (se o fator temporal assim permitir).

Chen et al. (2016) afirmaram que o limiar epidemiológico de uma rede é uma característica intrínseca de qualquer rede, isto é, toda rede tem o potencial de passar por uma epidemia. Para que ocorra uma epidemia é preciso que a força do vírus f seja maior que o limiar epidemiológico τ, portanto, se $f > \tau$, uma epidemia ocorrerá. De acordo com Prakash et al. (2012), em modelos cascata arbitrários, podemos estimar o limiar epidemiológico de uma rede utilizando o maior autovalor[2] da matriz de adjacência dessa rede como medida de vulnerabilidade. O grau de vulnerabilidade é a medida que busca quantificar o quão propensa uma rede é de, caso infectada por um vírus, uma epidemia acontecer. É importante ressaltar que existem diversas formas de avaliar a vulnerabilidade de uma rede (Musawi et al., 2023), o maior autovalor, também chamado de raio espectral, é somente uma delas. Representamos esse autovalor por λ_1, e definimos a seguinte relação (Chakrabarti et al., 2008): $\tau = \frac{1}{\lambda_1}$. Logo, quanto menor for nosso λ_1, maior é nosso τ, e menor é chance de ocorrer uma epidemia.

Em suma, é possível estimar a vulnerabilidade de uma rede utilizando seu raio espectral. Chen et al. (2016) argumentaram que a efetividade dessa medida é superior a outras medidas mais intuitivas, pois λ_1 está diretamente ligado com o número de caminhos e laços em um grafo. A fim de exemplificar essa característica, na Tabela 1.2 (Chen et al., 2016) temos alguns grafos e seus respectivos raios espectrais.

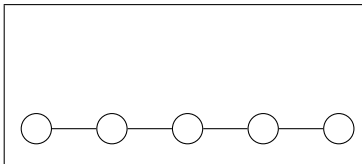
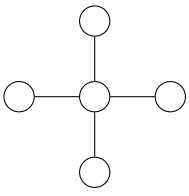
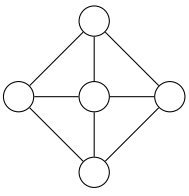
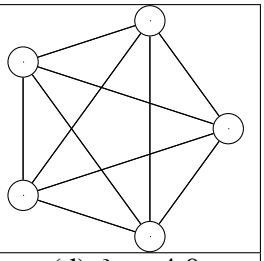
			
(a) $\lambda_1 = 1.7$	(b) $\lambda_1 = 2.0$	(c) $\lambda_1 = 2.9$	(d) $\lambda_1 = 4.0$

Tabela 1.2: Exemplo de medida de 'vulnerabilidade' em grafos. Mais arestas tornam o grafo mais bem conectado e, portanto, mais vulnerável. Note que a cadeia (a) e a estrela (b) têm o mesmo número de arestas, mas a medida λ_1 considera corretamente a estrela como mais vulnerável. Adaptado de Chen et al. (2016)

[2]Na literatura, esse autovalor é conhecido como **primeiro autovalor** de um grafo qualquer G

É importante notar que, intuitivamente, o grafo (a) é mais difícil de ter todos seus nodos infectados do que os demais grafos, ou seja, ele é o menos vulnerável. Perceba que o grau médio de (a) é o mesmo que de (b), mas (b) é mais vulnerável do que (a). O diâmetro do grafo (b) também é o mesmo de (c), porém este é mais vulnerável que o anterior. Em todos os casos, λ_1 captura corretamente o aspecto de vulnerabilidade de todos os grafos exemplificados – claro, essa não é uma prova cabal da efetividade da medida utilizada, mas atesta como ela pode ser pelo menos melhor que algumas outras medidas mais intuitivas, como o grau médio ou diâmetro.

1.4 O problema

Com as ferramentas bem definidas, agora é o momento de definir o problema e delinear estratégias para solucioná-lo. Em suma, buscamos imunizar entidades específicas em uma rede para minimizar o risco de uma epidemia, considerando um número limitado de recursos em comparação com o total de entidades na rede. Em termos computacionais, temos à nossa disposição k recursos para imunizar k vértices de um grafo G. O desafio reside em encontrar o melhor subconjunto S de G, de modo que a remoção desse subconjunto, denotado por $G - S = G'$, resulte na maior diferença possível entre o maior autovalor de G e G'. Essa diferença é conhecida como *"eigendrop"*, traduzida aqui livremente como autoqueda, e geralmente é utilizada como indicador de eficácia de imunização. É interessante notar que examinar a autoqueda ou $\lambda_1(G')$, quando conhecemos $\lambda_1(G)$, tem o mesmo efeito. Portanto, o uso do indicador autoqueda é justificado por ser unidimensional e bem estabelecido na literatura, com ressalvas: a instância (o grafo) e os recursos disponíveis (k) impactam diretamente na autoqueda. Por exemplo, remover nodos quaisquer de um grafo completo (todos são vizinhos de todos) pode resultar em uma autoqueda alta, mas não tornará a rede necessariamente mais imunizada, visto que qualquer nodo infectado poderá infectar todos os demais. Em suma, a autoqueda é uma medida escalar de imunização, mas não deve ser cegamente confiada: é preciso saber com qual grafo estamos lidando.

É importante lembrar que não é possível aumentar o maior autovalor de um grafo retirando vértices (Serre, 2002), portanto a remoção do conjunto S sempre tornará a rede menos vulnerável ou não afetará em nada na vulnerabilidade do sistema – a remoção de um nodo folha de um grafo árvore, por exemplo, não necessariamente contribui para tornar o sistema menos vulnerável.

Problema 1. *Dado um grafo simples não-direcionado e não-ponderado $G = (V, E)$, encontre um conjunto S de k vértices de modo que $\lambda_1(G - S)$ seja o mínimo possível sobre todas as escolhas de $S \subseteq V$.*

Com essa definição, torna-se evidente que o problema em questão é classificado como NP-Difícil (*NP-Hard*) quando k faz parte da entrada, como nesse caso, pois é possível realizar uma redução do problema para um problema de cobertura de vértices (Ahmad et al., 2017). Esse fato denota o nível de complexidade computacional em que o problema se enquadra, ou seja, é um problema de natureza exponencial, o que torna sua resolução ótima impraticável em termos de tempo, visto que é desconhecido um algoritmo eficiente que resolva o problema, com a melhor solução possível, em tempo hábil.

No entanto, isso não implica que o problema seja insolúvel: diversas soluções algorítmicas foram propostas, algumas das quais são abordadas neste trabalho. Todas essas estratégias podem ser categorizadas como Algoritmos de Aproximação, já que oferecem uma solução que, embora não seja ótima, é viável em termos de tempo e se aproxima, por um fator, da melhor solução possível. Nas seções seguintes, faremos uma breve revisão da literatura, apresentando os algoritmos candidatos junto com suas vantagens e limitações. O critério de

escolha dos algoritmos apresentados aqui foram: (1) Teoria Espectral dos Grafos é utilizada como ponto-chave do algoritmo de aproximação, (2) se tem menos de 10 anos e (3) se é um algoritmo de aproximação. Em seguida, são discutidas as implementações, acompanhadas de um resumo das características de cada algoritmo. Após essa etapa, temos os resultados de testes comparativos entre as estratégias. O trabalho é concluído com uma seção de discussão sobre o tema e sugestões para futuros trabalhos.

Capítulo 2

Revisão da Literatura

Na primeira década do século XXI vários pesquisadores abordaram o problema de propagação viral em redes. Pode-se dizer que Briesemeister et al. (2003) realizaram um dos primeiros estudos sobre propagação de vírus em redes de comunicação. Logo depois, Ganesh et al. (2005) estudaram os efeitos da topologia de um grafo no avanço de um epidemia e propuseram uma discussão sobre em que condições uma epidemia eventualmente termina sozinha, isto é, sem nenhum tipo de imunização. Contudo, é no trabalho de Chakrabarti et al. (2008), onde foi modelado uma forma de propagação viral em redes de comunicação, que os conceitos de limiar epidemiológico, taxa de nascimento e de morte de nodos e o uso do maior autovalor do grafo são relacionados – mais especificamente, a relação $\tau = \frac{1}{\lambda_1}$ é provada para grafos não-direcionados e sem peso.

Paralelamente, trabalhos em problemas similares foram desenvolvidos. O estudo da maximização de influência em redes sociais foi provado NP-Difícil, sobre certas circunstâncias, por Kempe et al. (2003). Além disso, técnicas similares à imunização de nodos foram utilizadas no problema de maximização de influência (Page et al., 1999; Rezaei et al., 2022; Liao et al., 2017). Apesar da similaridade com imunização, na prática as ideias são complementares: o problema da influência visa maximizar a influência (“infecção”) de algum vírus (informação, propaganda, opinião etc.) em uma rede, encontrando o conjunto de nodos mais influentes da rede. Enquanto a imunização visa minimizar essa característica, encontrando o conjunto de nodos que mais tornam a rede vulnerável. Além de outros estudos, como redundância de influência (Zhou et al., 2020) e mineração em grafos (Backstrom et al., 2006), que se assemelham com imunização de nodos – sendo até descritos de forma semelhante –, porém possuem características diferentes inerentes aos respectivos problemas.

No entanto, nenhum dos trabalhos supracitados propõe uma solução de imunização de nodos ou aborda o problema através dessa lente especificamente. Isso só foi realizado no trabalho de Chen et al. (2016), o que nos leva para nosso primeiro algoritmo abordado aqui.

2.1 NetShield e NetShield+

Proposto em Chen et al. (2016), o estudo nos traz algumas definições importantes para abordarmos o problema de forma algorítmica e aproximada, além de fornecer uma base teórica introdutória e testes relevantes para comparação entre metodologias aproximadas e exatas.

Para definir a vulnerabilidade de um grafo esse trabalho utiliza o maior autovalor da matriz de adjacência, como apresentado na introdução. Além disso, um *“shield-value”*, que aqui será traduzido livremente para valor-protetivo, é atrelado a um conjunto específico de nodos

através de uma heurística sub-modular, positiva e monotônica – o que garante que o cálculo desse valor-protetivo será escalável e aproximado.

Em outras palavras, cada conjunto possível de solução S, de tamanho k, terá uma pontuação, chamada de valor-protetivo, associada a ele. Essa pontuação é calculada utilizando uma heurística, i.e. um cálculo matemático que buscará aproximar, nesse caso, o quanto a remoção de S reduz $\lambda_1(G)$. O fato de essa heurística ser sub-modular[1] e monotônica[2] garante que a heurística será $(1 - \frac{1}{e})$-aproximada do ótimo (Nemhauser et al., 1978), sendo e o número neperiano. O valor protetivo é dado pela seguinte fórmula:

$$V_p(S) = \sum_{i \in S} 2 \cdot \lambda_1 \cdot u(i)^2 - \sum_{i,j \in S} A(i,j) \cdot u(i) \cdot u(j)$$

Nessa equação, S é o conjunto de vértices escolhido e Vp é o nome da função (valor-protetivo). λ_1 se refere à vulnerabilidade do grafo, u é o autovetor associado à λ_1, $u(i)$ e $u(j)$ são os auvetores das respectivas linhas (i) e colunas(j) da matriz de adjacência A do grafo.

A justificativa dessa fórmula pode ser encontrada em Chen et al. (2016) na seção 4.2 e foge do escopo desse trabalho demonstrar sua prova matemática. Portanto, falando apenas de maneira informal: esse valor associado a um conjunto S busca aproximar quanto de autoqueda esse conjunto produzirá quando removido da rede.

Apesar das definições fundamentadas de vulnerabilidade, valor-protetivo e da heurística, a principal contribuição desse estudo está na implementação do algoritmo NetShield e sua otimização NetShield+, que são apresentadas agora.

Algoritmo 1 NetShield

Entrada: A matriz de adjacência A e o número de recursos k
Saída: Um conjunto S de tamanho k a serem imunizados
Compute o primeiro autovalor λ de A; seja u seu autovetor correspondente $u(j)$ $(j = 1, \ldots, n)$;
Inicialize S como vazio;
for $j = 1$ **to** n **do**
    $v(j) = (2 \cdot \lambda - A(j,j)) \cdot u(j)^2$;
end for
for $iter = 1$ **to** k **do**
    Seja $B = A(:, S)$;
    Seja $b = B \cdot u(S)$;
    for $j = 1$ **até** n **do**
        if $j \in S$ **then**
            Seja $score(j) = -1$;
        else
            Seja $score(j) = v(j) - 2 \cdot b(j) \cdot u(j)$;
        end if
    end for
    Seja $i = \text{argmax}_j\ score(j)$, adicione i ao conjunto S;
end for
Return S

[1] Quantos mais itens são inseridos no conjunto de solução, menos o valor será importante inserir um novo item. Em outras palavras, os primeiros nodos a serem vacinados serão mais relevantes para o resultado final.

[2] É impossível diminuir o valor total da heurística adicionando elementos ao conjunto de solução. Em outras palavras, imunizar um nodo nunca tornará a rede menos imunizada do que antes.

Primeiro, calculamos o maior autovalor de A[3], depois seu vetor associado u[4]. Depois, calculamos o valor-protetivo de cada vértice e, em seguida, montamos o conjunto S que possui a maior soma possível dos valores-protetivos de cada vértice. Ao final teremos um conjunto S com os índices (nomes) dos vértices que resultaram no conjunto S de maior valor-protetivo. Mais detalhes sobre a implementação estão na seção 4.

A complexidade computacional do Algoritmo 1 é de $O(nk^2 + m)$, e a complexidade espacial é de $O(n + m + k)$ (Chen et al., 2016). Já sua otimização (Algoritmo 2) possui complexidade computacional de $O(nkb + mk/b)$, onde b é um parâmetro a mais explicado adiante. Sua complexidade espacial se mantém.

Algoritmo 2 NetShield+

1: **Entrada:** Uma matriz de adjacência A, dois inteiros k e b
2: **Saída:** Um conjunto S com k nodos
3: Calcule o número de iterações $t = \lfloor \frac{k}{b} \rfloor$
4: Inicialize S como vazio
5: **for** $j = 1$ **até** t **do**
6: Inicialize S' como vazio
7: $S' = \text{NetShield}(A, b)$
8: $S = S \cup S'$
9: Atualize A removendo os nós em S'
10: **end for**
11: **if** $k > tb$ **then**
12: $S' = \text{NetShield}(A, k - tb)$
13: $S = S \cup S'$
14: **end if**
15: **return** S

No NetShield+, introduzimos um novo parâmetro b, que determina o passo do processo de imunização. O objetivo é imunizar gradualmente a rede, selecionando alguns nós em cada iteração, imunizando-os e, em seguida, selecionando outros na próxima rodada. O número de iterações é determinado pelo quociente entre o número total de recursos disponíveis k e o passo b, arredondado para baixo. As linhas 11 a 14 do algoritmo são dedicadas a utilizar os recursos restantes, caso haja algum, já que o arredondamento para baixo pode resultar em uma subutilização dos recursos totais disponíveis. O trabalho original discute a relação entre o tempo de execução e a queda na eficácia do algoritmo ao modificar o valor de b. Um valor baixo de b resulta em uma eficácia melhorada, mas aumenta o tempo de execução do algoritmo. Por outro lado, aumentar b acelera a execução do algoritmo, mas reduz a eficácia da imunização. Em geral, argumenta-se que um valor de b em torno de $k/10$ oferece um bom equilíbrio entre desempenho e eficácia de imunização. No entanto, é importante ressaltar que, independentemente do valor de b, o resultado obtido continua sendo aproximado e o algoritmo possui complexidade de ordem polinomial de toda forma, o que configura a solução como aproximada e computacionalmente praticável.

Portanto, a escolha do NetShield e NetShield+ como candidato de comparação se dá devido à importância da contribuição desse trabalho para a literatura. Todos os demais candidatos citam, em algum nível, o trabalho explicado nessa subseção.

[3]Linha 9: A notação A(:,S) indica todas as linhas da matriz A indexadas pelos vértices do conjunto S
[4]Linha 10: uma projeção de u indexado pelos vértices de S

Um ano depois do NetShield, um grupo de pesquisadores, em 2017, buscaram trabalhar na heurística de nodos que mais aparecem em caminhos fechados no grafo, como a próxima seção explica.

2.2 Walk-4

Ahmad et al. (2017) realizaram uma extensa revisão bibliográfica para o problema de imunização de redes e complementaram o trabalho do NetShield através de conclusões adicionais sobre a heurística proposta pelo trabalho anterior. No estudo, foi descrito uma estrutura genérica de algoritmo guloso para imunização de nodos – Algoritmo 3.

Algoritmo 3 Guloso-Genérico

Entrada: Grafo G, recursos k
Saída: Conjunto S com k nodos
$S = \emptyset$
while $|S| < k$ **do**
    $v = \underset{v \in V \setminus S}{\operatorname{argmax}}(\lambda_1(A(G) - S \cup \{v\}))$
    $S = S \cup \{v\}$
end while
return S

Essa estrutura pode ser lida como: inicia-se um conjunto S. Enquanto o tamanho de S não for igual ao número de recursos k. Em cada iteração do laço, o índice do vértice que mais aumenta a autoqueda é selecionado e incluído no conjunto S. Ao fim, teremos um conjunto S com os vértices a serem imunizados. A ideia é similar ao NetShield e o algoritmo é polinomial, no entanto, como podemos saber qual vértice gera a maior autoqueda? O trabalho em questão propõe uma heurística baseada em contar qual vértice está na maioria de passeios fechados de tamanho 4 no grafo.

Um passeio por um grafo é um conjunto de arestas que indica como é possível ir de um dado vértice qualquer a até outro b. Um passeio fechado é quando um passeio começa e termina no mesmo vértice, podendo haver possibilidade de repetição de arestas. Um passeio de tamanho 4 a partir de um arbitrário vértice u até v, por exemplo, segue a notação (u, a, b, c, v), para que o passeio seja fechado temos $u = v$, isto é, (u, a, b, c, u).

Portanto, podemos reescrever o Algoritmo 3 da forma apresentada no Algoritmo 4

Algoritmo 4 Guloso-Passeio

Entrada: Grafo G, recursos k
Saída: Conjunto S com k nodos
$S = \emptyset$
while $|S| < k$ **do**
    $v = \underset{v \in V \setminus S}{\operatorname{argmax}} CW_4(G_S, v)$
    $S = S \cup \{v\}$
end while
return S

Dessa forma, o problema se remonta. Agora precisamos descobrir em quantos passeios fechados de tamanho 4 um dado vértice aparece. Começamos lembrando que é conhecimento

comum para a comunidade de estudiosos de grafos (West, 2001) que o número de caminhos fechados de tamanho p de um grafo qualquer pode ser obtido pelo cálculo do traço[5] da matriz de adjacência do grafo elevada a p. Ou seja:

$$CW_p(G) = tr(A(G)^p)$$

Logo, podemos realizar a seguinte operação algébrica para todo vértice em G,

$$CW_p(G) = CW_p(G - S, V \setminus S) + CW_p(G, S)$$

Dessa forma, notamos que o valor que queremos maximizar é dado por $CW_p(G, S)$, pois ele representa a diferença entre o número de caminhos de tamanho p do grafo original para o grafo produzido sem os vértices induzidos de S.

Apesar dessa formalização, o que temos até então é uma forma de descobrir a quantidade **total** de passeios fechados de tamanho p em uma rede. Contudo, precisamos estimar em quantos caminhos **cada vértice** possivelmente aparece. Para tanto, o trabalho original argumenta que podemos realizar uma aproximação desse valor, dada pela fórmula abaixo.

$$CW_4(G, v) = 2 \cdot d(v)^2 + 4 \cdot \sum_{\{u \in V, u \neq v\}} d(u, v)^2$$

Onde $d(v)$ é o grau[6] do vértice atual e $d(v, u)$ é o grau do vértice u, vizinho de v, subtraído de 1. Essa aproximação é explicada detalhadamente pelo trabalho em questão e foge do escopo deste estudo aprofundá-la. No entanto, notamos uma inconsistência com essa estimativa, que não foi abordada pelo trabalho original.

Observe o grafo exemplo a seguir, na Figura 2.1.

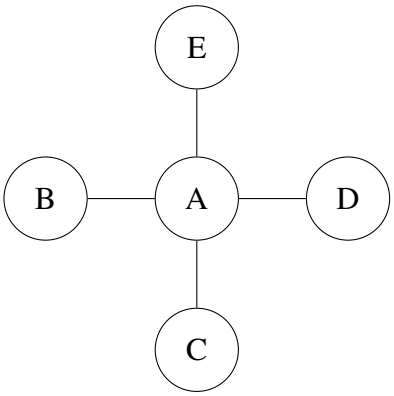


Figura 2.1: Grafo estrela.

Note que o vértice A é, intuitivamente, o vértice mais importante dessa rede para fins de imunização. Porém, se aplicarmos a estimativa proposta teremos uma pontuação incorreta para o nível de importância de cada vértice, como a conclusão abaixo demonstra.

$$CW_4(G, A) = 2 \cdot d(A)^2 + 4 \cdot \sum_{\{u \in V, u \neq v\}} d(u, A)^2 = 32 + 4 * 0 = 32$$

[5] corresponde à soma dos elementos da diagonal principal

[6] número de vizinhos

$$CW_4(G, B|C|E|D) = 2 \cdot d(B)^2 + 4 \cdot \sum_{\{u \in V, u \neq v\}} d(u, B)^2 = 2 + 4 * 9 = 38$$

Nesse caso, os vértices folhas, isto é, de grau 1, são erroneamente pontuados com muita superioridade em relação ao vértice central. Observando a prova de heurística em Ahmad et al. (2017), é possível notar que a fórmula acima parece funcionar corretamente só para grafos bem conectados (com números razoáveis de arestas) e não estima adequadamente grafos estrelas ou pouco conectados. Isso ocorre porque o número 4 surge das possíveis posições que um vértice pode aparecer, pelo menos uma vez, nos conjuntos que descrevem caminhos de tamanho 4 no dado grafo. Por exemplo, o caminho (A, B, C, D, A) é contabilizado como possível na estimativa, mas claramente ele é impossível no grafo estrela, pois não há aresta que ligue B com C. Na verdade, a única forma de existir um caminho fechado de tamanho 4 partindo do vértice central é se o mesmo aparecer no meio do caminho, isto é, (A, B, A, C, A). Já se não partimos de A, teremos A obrigatoriamente na segunda e quarta posição, como (C, A, D, A, C) nos mostra. Vale lembrar que o grafo estrela possui uma simetria muito útil aqui, pois facilita a demonstração para o grafo todo e ainda mantém a escalabilidade dessa constatação. Portanto, a heurística proposta superestima o cograu[7] de v, pois é muito otimista com a conectividade do grafo.

Na próxima seção apresentamos uma heurística alternativa que abarcará os casos em que a heurística original falha. Por ora, contudo, continuemos com a aplicação da heurística proposta em conjunto com o algoritmo genérico 4.

Os algoritmos auxiliares que computam a pontuação e a atualização a cada iteração estão descritos, respectivamente, pelo algoritmo 5 e 6.

Algoritmo 5 COMPUTA-PONTUAÇÃO

Entrada: Grafo G
Saída: Vetor de pontuações
Inicialize $codeg_sum[v] = 0$ para todos os vértices $v \in G$
Inicialize $score[v] = 0$ para todos os vértices $v \in G$
Inicialize $degrees[v] = d_g(v)$ para todos os vértices $v \in G$
for $nodo \in G$ **do**
    for $vizinho \in N(nodo)$ **do**
        $codeg_sum[vizinho] \leftarrow codeg_sum[vizinho] + (degrees[nodo] - 1)$
    end for
end for
for $nodo \in G$ **do**
    $score[nodo] \leftarrow 2 \times degrees[nodo]^2 + 4 \times codeg_sum[nodo]^2$
end for
return $score$

[7] soma dos graus de todos os vizinhos do vértice em questão

Algoritmo 6 ATUALIZA-PONTUAÇÃO

Entrada: Grafo G, nodo
Saída: Vetor de pontuações
for $vizinho \in G$ **do**
    $degrees[vizinho] \leftarrow degrees[vizinho] - 1$
    $codeg_sum[vizinho] \leftarrow codeg_sum[vizinho] - (degrees[nodo] - 1)$
    for $vizinho_do_vizinho \in N(vizinho)$ **do**
        $codeg_sum[vizinho_do_vizinho] \leftarrow codeg_sum[vizinho_do_vizinho] - 1$
    end for
end for
$degrees[nodo] \leftarrow 0$
$codeg_sum[nodo] \leftarrow 0$
$score[nodo] \leftarrow 0$
$vizinhos \leftarrow N(nodo)$
$G.remove_nodo(nodo)$
for $vizinho \in vizinhos$ **do**
    $score[vizinho] \leftarrow 2 \times degrees[vizinho]^2 + 4 \times codeg_sum[vizinho]^2$
    for $viz_do_viz \in N(vizinho)$ **do**
        $score[viz_do_viz] \leftarrow 2 \times degrees[viz_do_viz]^2 + 4 \times codeg_sum[viz_do_viz]^2$
    end for
end for
return $score$

No algoritmo 5 inicializamos os vetores que utilizamos. Em seguida, calculamos o cograu de cada vértice e calculamos o score de cada um desses vértices de acordo com a heurística correspondente. A complexidade temporal do algoritmo é $O(m)$, como demonstrada pelo **lemma 4** da seção 4.2 em Ahmad et al. (2017).

Já no algoritmo 6 primeiro são feitos ajustes decorrentes da eventual remoção do nodo passado como argumento para a função. Em seguida, removemos o nodo do grafo e inutilizamos sua influência para a pontuação a ser recalculada. Por fim, calculamos a nova pontuação para todos os vértices impactados – vizinhos e vizinhos de vizinhos – pela remoção supracitada. A complexidade desse algoritmo é $O(m)$.

Com isso, temos dois algoritmos lineares que nos auxiliam a calcular uma pontuação. Agora, para contemplarmos todas as peças de nosso quebra-cabeça ficamos com o algoritmo 7, que possui complexidade computacional de $O(n + km + klog(n))$ e complexidade espacial de $O(m + n + k)$ quando usamos uma MAX-HEAP para acomodar o vetor de pontuações. Vale lembrar que não é do escopo deste trabalho provar a corretude ou a complexidade dos algoritmos aqui citados.

Algoritmo 7 Walk-4

Entrada: Grafo G, recursos k
Saída: Conjunto S com k nodos
$S = \emptyset$
COMPUTA-PONTUAÇÃO(G)
while $|S| < k$ **do**
    $v = \underset{u \in V \setminus S}{\text{argmax}}(score'_G[u])$
    $S = S \cup \{v\}$
    ATUALIZA-PONTUAÇÃO(G, v)
end while
return S

2.3 Non-backtracking X-Centrality (X-NB)

Por fim, veremos o estudo de centralidade de nodo, que busca valorar os nodos de uma rede a fim de descobrir qual vértice é o mais central, geralmente classificado como mais importante para fins de maximizar autoqueda.

O conceito de centralidade e "nodo central" não é objetivo, isto é, existem diversas formas de definir essas características. Algumas delas são a centralidade por grau, onde os vértices de maior grau são tidos como mais centrais; outra é por proximidade, onde são usados os menores caminhos de um vértice aos demais vértices do grafo como parâmetro para medir centralidade; centralidade intermediária (*betweenness*) busca medir quantas vezes um nodo atua como uma ponte entre os menores caminhos entre outros nodos; centralidade por autovetores (ou autocentralidade (*eigencentrality*)) mede o quão influente um nodo é em uma rede – o algoritmo PageRank (Page et al. (1999)), descrito como o início do algoritmo de busca do Google, é uma variação desse tipo de centralidade –, entre outras medidas.

O que nos interessa aqui é a medida de centralidade estudada por Torres et al. (2020), que propõe uma heurística baseada na matriz de não-retrocesso (*Non-backtracking*) da rede analisada.

Primeiro, é importante reiterar que diversas matrizes, além da matriz de adjacência, podem ser geradas a partir de um grafo e que diferentes estudos espectrais são realizados nelas. A título de exemplo, a matriz LaPlaciana também pode prover formas de medir vulnerabilidade de rede (Zhao et al., 2024), apesar de ser calculada de forma completamente diferente da matriz de adjacência e ainda possuir outras características matriciais. Portanto, vamos definir o que é uma matriz de não-retrocesso segundo o próprio artigo apresentado aqui.

Uma matriz de não-retrocesso B é formada sempre que temos um passeio não-retrocedente, que consiste em um passeio em que não há retornos, por parte do "caminhador", pela mesma aresta. Por exemplo, o passeio (a, b, c, b) é retrocedente, pois o caminhador vai de b para c e volta, pela mesma aresta, para b. A definição não formal seria um passeio "em que só se anda para frente"; outra forma de visualizar isso seria classificando os passeios: um passeio é dito não-retrocendente (*non-backtracking walks*, ou *NB-walk*), se nenhuma aresta consecutiva forma um retrocesso.

Armados dessa definição, entendemos uma NB-matriz (NB sendo a sigla para *non-backtracking*) com o delta de Kronecker δ, da seguinte forma:

$$B_{k\to l, i\to j} = \delta_{jk}(1 - \delta_{il})$$

Que pode ser lida da seguinte forma: $B_{k\to l,i\to j}$ é 1 se, e somente se, $j = k$ e $i \to j, j \to l$ não é um retrocesso; nos demais casos será 0.

O objetivo por trás dessa matriz é obter uma noção de centralidade melhor que a autocentralidade (Martin et al., 2014; Radicchi e Castellano, 2016), utilizada em algoritmos como o PageRank. Agora, para a obtenção dessa matriz, são necessários vários passos descritos e justificados no artigo original. Introduzi-los ou explicá-los aqui excederia o escopo deste trabalho, pois envolvem conceitos avançados de álgebra linear e operações matriciais, culminando em um método rápido para calcular a centralidade de um nodo a partir da matriz de não-retrocesso. O que é importante sabermos é que o método utilizado possui dois obstáculos a serem vencidos pelos algoritmos propostos:

i) A centralidade X-NB de um nodo deve ser calculada **após** a remoção desse nodo. Portanto, para que possamos decidir qual nodo deve ser removido a cada iteração do nosso algoritmo de aproximação, precisamos remover temporariamente cada um dos vértices para, somente então, definir qual deve ser removido em última instância. Isso mataria o propósito da imunização aproximada – perceba que seria o mesmo que testar todas as possibilidades, pois teríamos que remover os vértices um a um e calcular um valor para os grafos gerados pós-remoção.
ii) Para imunizar apropriadamente a rede, precisaremos sempre recomputar a pontuação (centralidade) de cada nodo do grafo após a remoção definitiva de um nodo. Esse processo precisa ser eficiente.

Para resolver o problema respeitando as duas restrições descritas anteriormente podemos implementar o seguinte algoritmo 8.

Algoritmo 8 X-NB Aproximação

Entrada: Grafo G, inteiro p
Saída: Lista ordenada $removed$ de nodos para imunizar
$removed = \emptyset$
$XNB[i] = 0$ para cada nodo i
while $|removed| < p$ **do**
  v_G = principal autovetor de $AuxNBMatrix(G)$
  for cada nodo c em G **do**
    $XNB[c] = XNBCentrality(v_G, c)$
  end for
  $node = \text{argmax}_i(XNB[i])$
  $G = RemoveNode(G, node)$
  $removed = removed \cup \{node\}$
end while
return $removed$

A ideia aqui é calcular a matriz auxiliar de não-retrocesso. Na prática, o uso da matriz auxiliar é melhor para o cálculo da centralidade do que a matriz principal, pois B_{aux} é de tamanho $2n \times 2n$, enquanto B é de tamanho $2m \times 2m$. Para se obter B_{aux} precisamos da matriz identidade I de tamanho n; a matriz de adjacência de G e a matriz diagonal D com os graus dos vértices na diagonal. Dessa forma temos o seguinte:

$$B_{aux} = \begin{bmatrix} 0 & D - I_n \\ -I_n & A \end{bmatrix}$$

Enfim, obtendo o principal autovetor da matriz auxiliar de G, conseguimos calcular o valor da centralidade de cada nodo. Esse cálculo é feito de maneira aproximada e está

detalhadamente descrito no capítulo 3.2.1 do trabalho em questão (Torres et al., 2020). Não é importante para nós entrar nos detalhes desse cálculo, pois sua natureza é muito mais do campo da álgebra linear do que da aproximação de algoritmos ou imunização de redes. Entender que é a partir daqui que temos um ordenamento entre os nodos já é o bastante. Por fim, removemos o nodo de maior centralidade e prosseguimos até gastarmos todas nossas vacinas. A complexidade desse algoritmo é $O(k(m+n))$.

Assim, finalmente, cobrimos todos os algoritmos escolhidos nesse trabalho a partir da literatura.

Capítulo 3

Walk-4 Aprimorado

Como comentado no capítulo anterior, há um problema com a heurística estabelecida no algoritmo Walk-4: ela estima erroneamente a quantidade de caminhos possíveis quando o grafo é pouco denso. Para analisarmos a questão vamos lançar mão de alguns grafos, exemplo:

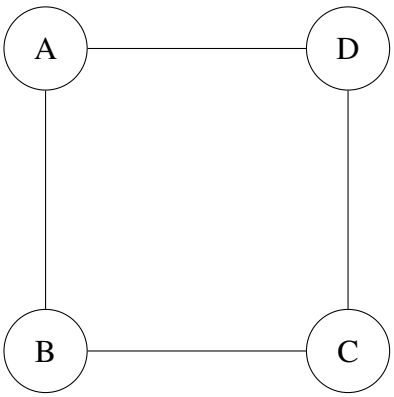


Figura 3.1: Ciclo 4

Na Figura 3.1 vemos que a heurística funciona apropriadamente, pois o vértice A pode estar em quaisquer posições possíveis em passeios fechados de tamanho 4. Perceba que o mesmo vale para os demais vértices devido à simetria do grafo em questão. Agora, se removermos o nodo C, ficamos com o grafo:

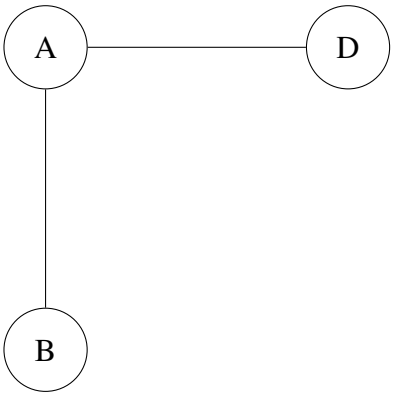


Figura 3.2: Ciclo 4 sem o vértice C

Percebemos, então, que na figura 3.2 o vértice A se encontra como centro de uma estrela, logo, a heurística avaliará erroneamente essa situação, como demonstrado anteriormente.

Portanto, é possível notar que a condição para que o vértice em questão apareça nas 4 posições de um passeio fechado é o número de vértices a distância 2 do nodo em questão. Em outras palavras, o nodo B na figura 3.2 possui o vértice D a dois vértices de distância, se calcularmos os passeios fechados que contém B teremos: (B,A,D,A,B), (B,A,B,A,B), (A,B,A,D,A), (A,D,A,B,A), (D,A,B,A,D) e (A,B,A,B,A). Notamos que B aparece em todas as posições possíveis, por possuir D a uma distância 2 de si. Já o vértice A, se aparecer na primeira posição, teremos-o, obrigatoriamente, na segunda e na última posição (afinal o caminho é fechado); se A está na segunda posição, então, estará também na quarta, pois não há vértices à 2 de distância de si – i.e. não há vizinhos de vizinhos para A. Com isso, fica possível verificar que basta que tomemos medidas para não superestimar o cograu dos vértices que estão nas folhas dessas estruturas, ou seja, basta que chequemos o grau dos vértices antes de atribuirmos pontos para eles. Nesse caso, B e D não devem ser considerados com uma pontuação significativa, pois eles se sobressairão em relação ao vértice A, que deveria ter uma pontuação maior. Essa conclusão também vai de encontro com a intuição, uma vez que nodos de grau 1, ou seja, folhas, não são bons candidatos para imunização, pois contribuem muito pouco para tornar o grafo mais seguro.

Agora, basta que mudemos o algoritmo 7 de modo a acrescentar essa verificação, como está nos algoritmos 9 e 10.

Algoritmo 9 COMPUTA-PONTUAÇÃO-APRIMORADO

1: **Entrada:** Grafo G, Vetores $degrees$, $codeg_sum$ e $score$
2: **Saída:** Vetor de pontuações $score$
3: **for** $nodo \in G.nodes()$ **do**
4: **for** $vizinho \in G.neighbors(nodo)$ **do**
5: $codeg_sum[vizinho] \leftarrow codeg_sum[vizinho] + (degrees[nodo] - 1)$
6: **end for**
7: **end for**
8: **for** $nodo \in G.nodes()$ **do**
9: **if** $degrees[nodo] \neq 1$ **then**
10: $score[nodo] \leftarrow 2 \times degrees[nodo]^2 + 4 \times codeg_sum[nodo]^2$
11: **end if**
12: **end for**
13: **return** $score$

Algoritmo 10 ATUALIZA-PONTUAÇÃO-APRIMORADO

Entrada: Grafo G, Nodo $nodo$, Vetores $degrees$, $codeg_sum$ e $score$
Saída: Vetor de pontuações $score$
for $vizinho \in G.neighbors(nodo)$ **do**
    $degrees[vizinho] \leftarrow degrees[vizinho] - 1$
    $codeg_sum[vizinho] \leftarrow codeg_sum[vizinho] - (degrees[nodo] - 1)$
    for $vizinho_do_vizinho \in G.neighbors(vizinho)$ **do**
        $codeg_sum[vizinho_do_vizinho] \leftarrow codeg_sum[vizinho_do_vizinho] - 1$
    end for
end for
$degrees[nodo] \leftarrow 0$
$codeg_sum[nodo] \leftarrow 0$
$score[nodo] \leftarrow 0$
$vizinhos \leftarrow \text{list}(G.neighbors(nodo))$
$G.remove_node(nodo)$
for $vizinho \in vizinhos$ **do**
    if $degrees[vizinho] == 1$ **then**
        $score[vizinho] \leftarrow 0$
    else
        $score[vizinho] \leftarrow 2 \times degrees[vizinho]^2 + 4 \times codeg_sum[vizinho]^2$
    end if
    for $viz_do_viz \in G.neighbors(vizinho)$ **do**
        if $degrees[viz_do_viz] == 1$ **then**
            $score[viz_do_viz] \leftarrow 0$
        else
            $score[viz_do_viz] \leftarrow 2 \times degrees[viz_do_viz]^2 + 4 \times codeg_sum[viz_do_viz]^2$
        end if
    end for
end for
return $score$

Apesar de simples, essa mudança garante que os casos anteriores sejam devidamente calculados. Outra forma de analisar essa heurística seria calculando quantos vizinhos de vizinhos cada nodo tem, o que poderia ser feito através da matriz de adjacência do grafo original elevada ao quadrado (essa operação culmina em uma matriz que tem, em sua diagonal principal, a quantidade de caminhos de tamanho 2 de cada vértice (West, 2000)). No entanto, isso adicionaria dois problemas na solução final: o primeiro seria a sobrecarga de cálculo e atualização dessa estrutura que guarda o número de caminhos entre dois nodos, pois cada nodo removido afeta a sua vizinhança e a vizinhança dos seus vizinhos, adicionando um custo computacional da ordem cúbica (Golub e Van Loan, 1996), pois no mínimo estamos executando uma multiplicação de matrizes. Na prática, a atualização da quantidade de vizinhos e vizinhos de vizinhos poderia ser paralelizada e, também, dependeria do grau do nodo que está sendo removido, afinal a vizinhança pode ter tamanho variado. Portanto, possivelmente, o algoritmo final poderia ser rápido, mas a sobrecarga de operações ainda o deixaria teoricamente mais lento que o algoritmo apresentado anteriormente, e possivelmente não atingiria resultados muito melhores, visto que aumentar o tamanho dos passeios fechados é mais efetivo (Tariq et al., 2017).

Capítulo 4

Implementações

Os algoritmos que serão comparados aqui são:

- BruteForce
- NetShield e sua otimização NetShield+
- Walk-4
- Walk-4 Aprimorado
- Non-backtracking X-centrality (X-NB)

Todos foram implementados em Python. O algoritmo Walk-4 teve suas variações, walk-4 e walk-8, fornecidas por Tariq et al. (2017) e foram feitos em MATLAB. A implementação do X-NB (Torres et al., 2020) foi gentilmente fornecida por Leo Torres, um dos principais autores do artigo original. As implementações podem ser encontradas em Heise (2024).

4.1 BruteForce

Algoritmo padrão de força bruta, testa todas as possibilidades de subconjuntos de vértices S de tamanho k a serem removidos de G, guarda o valor e o conjunto que resultou na maior autoqueda e o retorna.

As características gerais do algoritmo podem ser vistas na tabela 4.1. As complexidades são provadas por Ahmad et al. (2017).

Complexidade computacional	$O(\binom{n}{k}m)$
Complexidade espacial	$O(\binom{n}{k})$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.1: Características do BruteForce

A implementação está apresentada no Algoritmo 11, em pseudocódigo:

Algoritmo 11 BruteForce(G, k)

```
1: S = ∅
2: maior_queda = −∞
3: for cada subconjunto S' de tamanho k em G.nodos do
4:     queda = autoqueda(G, S')
5:     if queda > maior_queda then
6:         maior_queda = queda
7:         S = S ∪ S'
8:     end if
9: end for
10:
11: return S, maior_queda
```

Vale ressaltar que esse algoritmo **não será usado nos testes comparativos**, pois sua complexidade exponencial impossibilita a realização de testes extensivos em grafos grandes. Esse algoritmo foi utilizado somente para avaliar, em grafos pequenos, a corretude das heurísticas que são utilizadas nos demais códigos. Na seção de testes esse assunto é contemplado com mais detalhes.

4.2 NetShield e NetShield+

A implementação foi direta e fiel ao artigo referência.
A tabela 4.2 contém informações gerais sobre essa implementação do algoritmo 1.

Complexidade computacional	$O(nk^2 + m)$
Complexidade espacial	$O(n + m + k)$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.2: Características do NetShield

Já a tabela 4.3 resume as características gerais do algoritmo 2 (NetShield+).

Complexidade computacional	$O(mk/b + nkb)$
Complexidade espacial	$O(n + m + k)$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.3: Características do NetShield+

A complexidade espacial se mantém a mesma nos dois algoritmos, já a complexidade computacional é potencialmente melhorada se b for significativamente menor que k. Afinal, se $b = k$, temos:

$$mk/b + nkb = mk/k + nkk = m + nk^2$$

Percebemos, então, que nessas condições ficamos com a complexidade do algoritmo original. Para fins práticos, utilizar b como $k/10$ tem a melhor relação precisão-complexidade estatisticamente (Chen et al., 2016).

4.3 Walk-4

A implementação do Walk-4 original prevê o uso de uma max-heap para armazenar o valor de pontuação de cada nodo em cada rodada, o que garante uma complexidade de $O(n)$ para ser construída e $O(logn)$ para atualizar um valor na heap ou para ler o maior valor armazenado. Em outras palavras, utilizando uma max-heap o algoritmo terá uma complexidade de $O(n+km+klogn)$, pois os valores são lidos e atualizados k vezes. Contudo, na implementação feita neste trabalho, foi utilizado um simples vetor para armazenar os valores de pontuação; logo, a complexidade da atualização e busca é linear, ou seja, $O(n)$, portanto substituímos o $logn$ por n, e ficamos com $(O(n+km+kn)$. As características gerais desta implementação estão descritos pela tabela 4.5.

Complexidade computacional	$O(n + km + kn)$
Complexidade espacial	$O(n + m)$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.4: Características do Walk-4

4.4 Walk-4 Aprimorado

O Walk-4 aprimorado possui a mesma complexidade do Walk-4, com uma checagem a mais para garantir que o algoritmo cobrirá casos como grafo estrela e grafos pouco densos em geral. A implementação segue a mesma lógica do algoritmo Walk-4.

Complexidade computacional	$O(n + km + kn)$
Complexidade espacial	$O(n + m)$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.5: Características do Walk-4

4.5 X-NB

A implementação do algoritmo 8 foi disponibilizada por um dos autores do artigo de referência, portanto, está exatamente igual ao artigo original. Suas características gerais estão definidas a seguir pela tabela 4.6.

Complexidade computacional	$O(k(m+n))$
Complexidade espacial	$O(m+n)$
Parâmetros	$\{G, k\}$
Retorno	$\{S\}$

Tabela 4.6: Características do X-NB

4.6 Resumindo

Nesta subseção, consolidamos as informações de todas as implementações discutidas anteriormente. A Tabela 4.7 apresenta um resumo abrangente, facilitando a comparação das características e complexidades dos diferentes algoritmos. É importante ressaltar que a complexidade espacial de todos, menos o BruteForce, é dada por $O(n+m+k)$, mas como k é muito pequeno em relação à n, e não sabemos se o grafo terá m e n da mesma grandeza, optamos por deixar todos como $O(n+m)$.

Algoritmo	Complexidade computacional	Complexidade espacial	Parâmetros	Retorno
BruteForce	$O(\binom{n}{k}m)$	$O(\binom{n}{k})$	$\{G, k\}$	$\{S\}$
NetShield	$O(nk^2+m)$	$O(n+m)$	$\{G, k\}$	$\{S\}$
NetShield+	$O(mk/b+nkb)$	$O(n+m)$	$\{G, k\}$	$\{S\}$
Walk-4	$O(n+km+kn)$	$O(n+m)$	$\{G, k\}$	$\{S\}$
Walk-4 Aprimorado	$O(n+km+kn)$	$O(n+m)$	$\{G, k\}$	$\{S\}$
X-NB	$O(k(m+n))$	$O(n+m)$	$\{G, k\}$	$\{S\}$

Tabela 4.7: Características dos diferentes algoritmos

Capítulo 5

Testes

5.1 Método

Todos os testes foram executados em um computador com as seguintes características:

- **Sistema Operacional:** EndeavourOS Linux x86_64
- **Kernel:** 6.6.41-1-lts
- **CPU:** Intel i7-6700 (8) @ 4.000GHz
- **GPU:** Intel HD Graphics 530
- **Memória:** 15670 MiB

Os grafos foram construídos na linguagem DOT (Graphviz, 2024). A implementação de todos os algoritmos foi em python, na sua versão 3.12.4. Todos os grafos utilizados podem ser encontrados no repositório deste trabalho (Heise, 2024) na pasta *entradas*. Dois dos grafos utilizados (gen_graph e gen_tree) foram gerados através do gerador para olimpíadas de programação, jngen, disponível em ifsmirnov (2024). O grafo netshield foi tirado do artigo do NetShield (Chen et al., 2016). O grafo seguidores foi tirado a plataforma integrada MECRED (C3SL, 2024), um projeto do laboratório de pesquisa C3SL (Centro de Computação Científica e Software Livre). Trata-se de um grafo de rede sociais que representa, em seus vértices, os usuários da plataforma e as arestas são as relações de quem segue/é seguido por quem. Os demais grafos, web-spam e tech-WHOIS, foram tirados do repositório de Rossi e Ahmed (2015). O grafo web-spam indica as interações entre hyperlinks de spams na internet, já o grafo tech-WHOIS não possui metadados oficiais, mas é possível interpretá-lo como o grafo de interações entre domínios e sites gerados pelo comando whois.

Todos os algoritmos foram rodados com diferentes quantidades de recursos k, calculada como uma função de n. Incrementalmente, k começa como $n/20$, depois $n/16$, $n/12$, $n/8$ e, por fim, $n/4$. Vale ressaltar que, geralmente, temos muito menos vacinas do que entidades vacináveis (Sparrow et al., 2021), portanto, k costuma ser pequeno em relação à n. Dessa forma, podemos observar a escalabilidade de cada algoritmo conforme maior é S.

A seguir a tabela 5.1 resume os grafos utilizados para os testes.

Nome	Vértices	Arestas	Descrição
gen_graph	50	100	Grafo gerado aleatoriamente pelo gerador jngen.
gen_tree	50	100	Grafo árvore gerado aleatoriamente pelo gerador jngen.
netshield	16	15	Grafo encontrado no artigo do NetShield.
seguidores	2536	3921	Grafo de seguidores do MECRED.
web-spam	4767	37375	Grafo disponível no repositório online. Hyperlink de spams que levam para outros sites.
tech-WHOIS	7476	56943	Grafo disponível no repositório online. Não há descrição correta dos metadados.

Tabela 5.1: Grafos utilizados para os testes.

5.2 Gráficos

Para fins de detalhe, vamos analisar quais nodos cada algoritmo imuniza no grafo netshield, Figura 5.1.

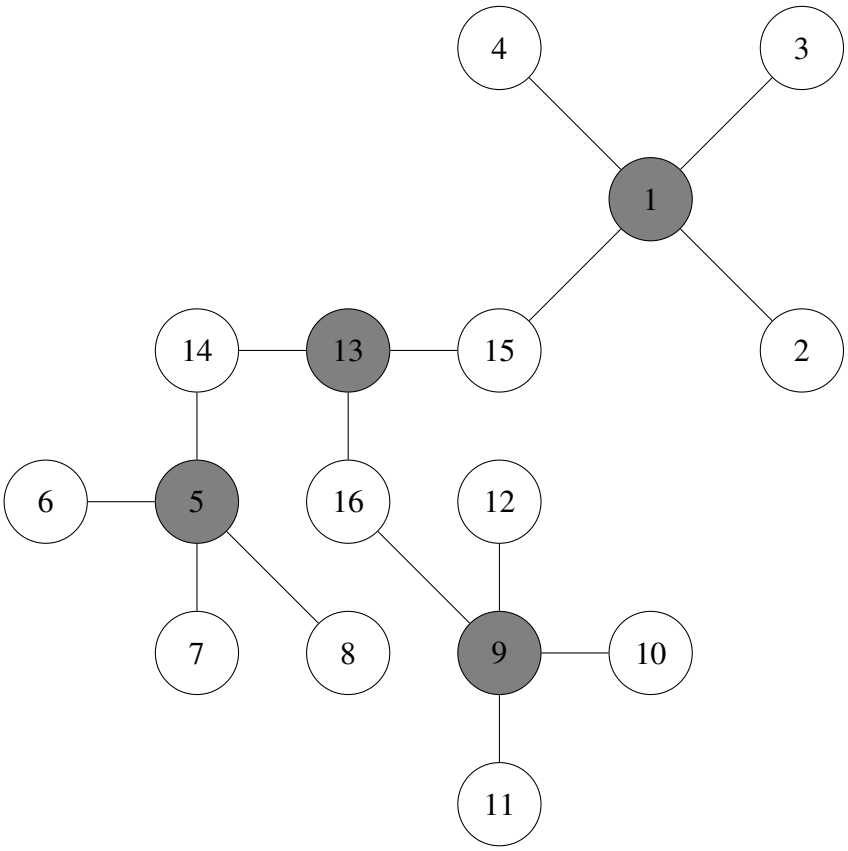


Figura 5.1: Representação do grafo netshield. Em cinza estão os nodos imunizados pelo algoritmo 1. Adaptado de Chen et al. (2016)

Nesse grafo, percebemos que com 4 vacinas somos capazes de imunizar o grafo da melhor forma possível: removendo os vértices 9, 5, 1 e 13. O algoritmo 1 faz exatamente isso. Já os algoritmos Walk-4 e Walk4-aprimorado imunizam de maneira igual: os vértices 1, 15, 14 e 16 são escolhidos pelas suas heurísticas. A ordem, no entanto, muda de algoritmo para algoritmo; Walk-4 imuniza primeiro o vértice 16, depois 15, 14 e por fim o 1. Já sua versão aprimorada imuniza os vértices 15, 1, 16 e 14, como está na figura 5.2. De toda forma, para essa árvore em específico, a imunização se mostra sub-ótima com o número ótimo de vacinas. O mesmo acontece com o X-NB, que imuniza os vértices 13, 14, 15 e 9, como está na figura 5.3.

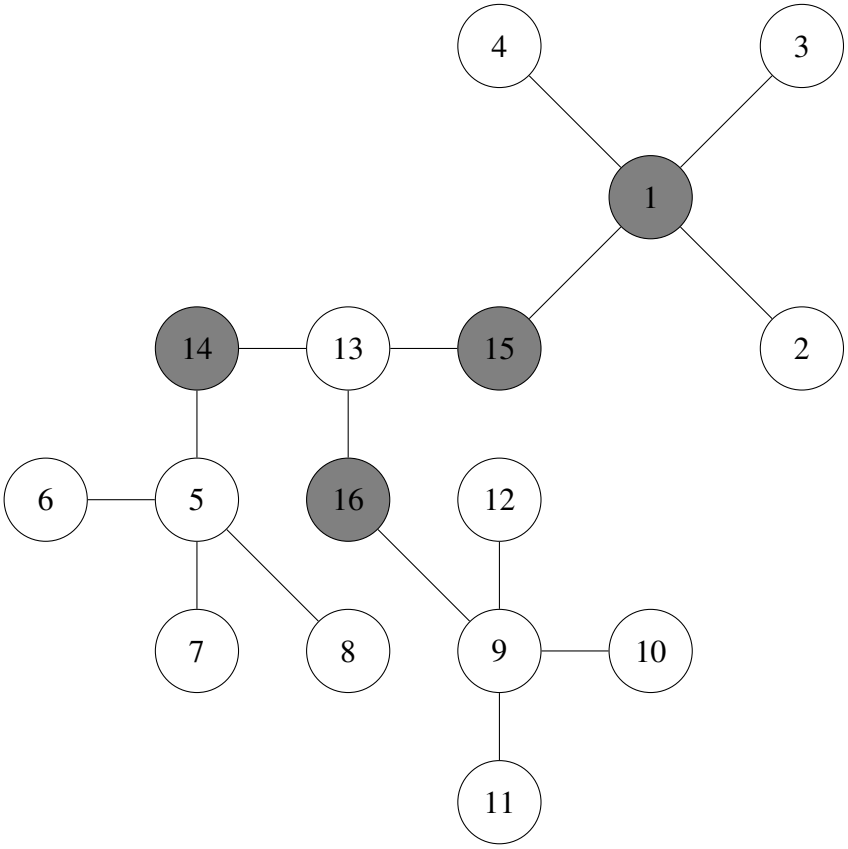


Figura 5.2: Representação do grafo netshield imunizado pelo Walk-4 e Walk4-aprimorado.

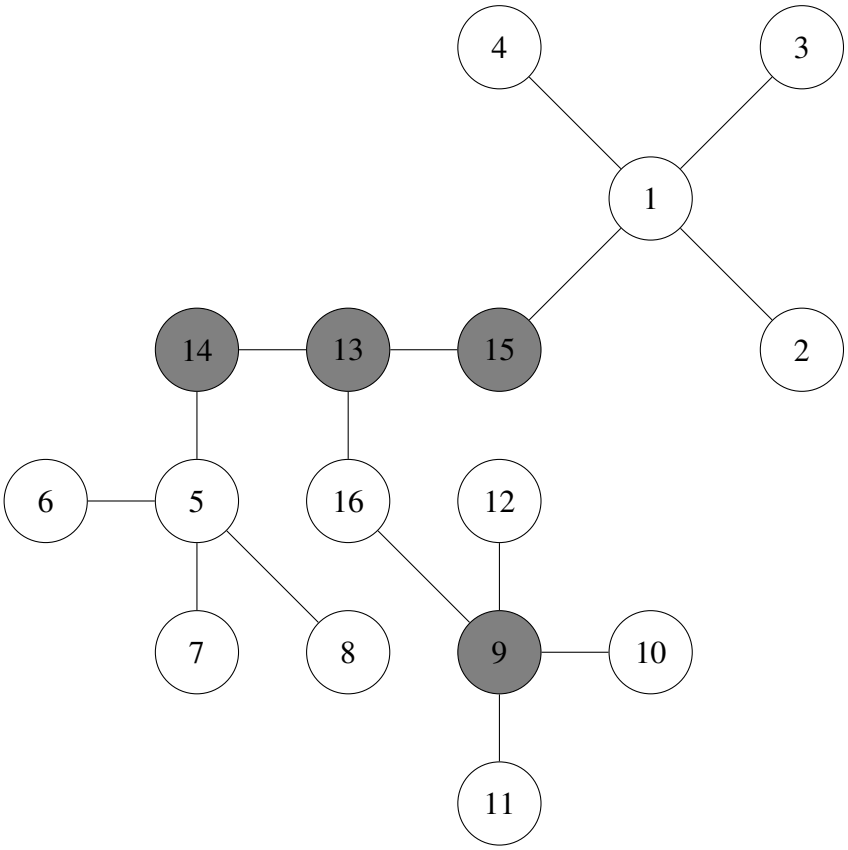


Figura 5.3: Representação do grafo netshield imunizado pelo X-NB.

No entanto, esse grafo pode nos enganar: o problema de imunização é focado para redes grandes, já que para redes pequenas o problema seria possivelmente simples de resolver por humanos, ou utilizando estratégias combinatórias.

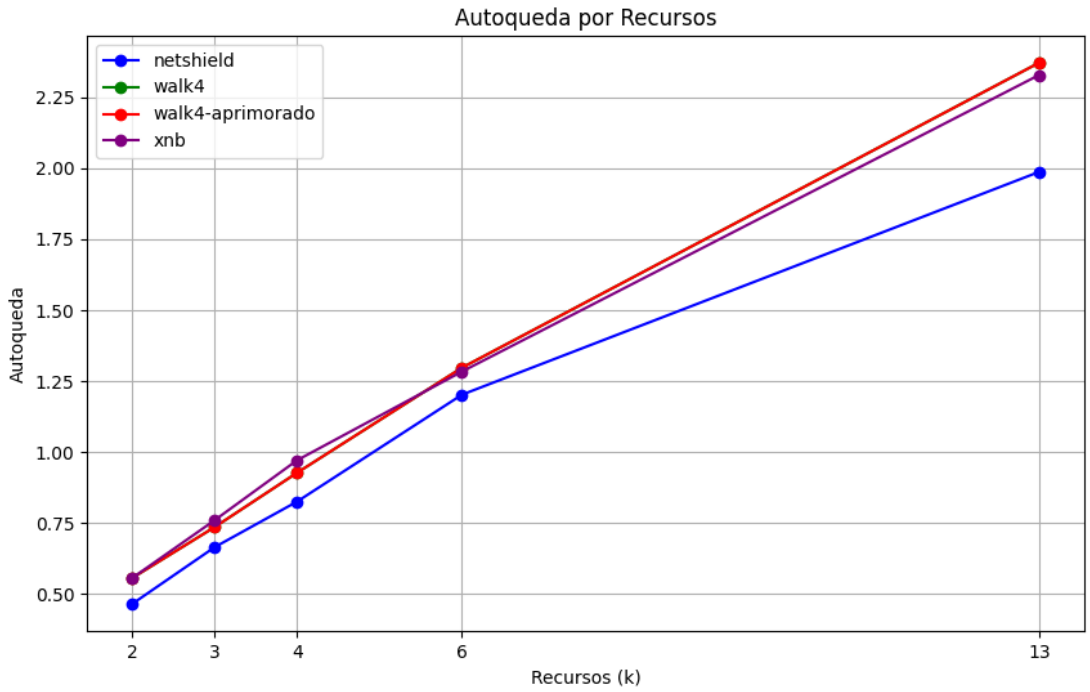


Figura 5.4: Gráfico de autoqueda por número de recursos para o grafo gen_graph, 50 vértices e 100 arestas.

Para os 2 grafos gerados artificialmente notamos que a linha verde, Walk-4 (algoritmo 7, não aparece. Isso acontece porque ela é concorrente ao Walk-4 Aprimorado (algoritmo 3), ou seja, produziram os mesmos resultados. Ademais, é possível notar que o Walk-4 Aprimorado pontua melhor para 13 recursos com um dos menores tempos de execução.

5.3 Resultados

Com todos os gráficos, podemos aferir que o X-NB costuma ser muito mais lento que os demais algoritmos. Através do *profiling* do código, foi identificado que a função *scipy.sparse.linalg.eigs()* é a responsável pela demora de execução, o que parece correto, já que ela calcula – de forma aproximada – as entradas do principal autovetor esquerdo da matriz auxiliar de NB. Esse calculo toma bastante tempo e, muitas vezes, não converge. Aumentando o parâmetro nvc, que é o número de autovetores auxiliares de Laczos, conseguimos fazer com que a conversão ocorra para grafos grandes a troco do desempenho, em tempo de relógio, do programa.

Em diversos casos, Walk-4 e Walk-4 Aprimorado resultaram na mesma autoqueda e em um tempo de execução muito similar. Em alguns casos, no entanto, como na figura 5.14 e na figura 5.10, vemos uma expressiva vantagem do algoritmo aprimorado para o algoritmo padrão. Especialmente no grafo de seguidores. Acreditamos que essa variação ocorre devido ao grafo de seguidores ser bastante mal comportado: ele possui vários componentes não-conexos de tamanhos extremamente variados e muitos nodos de grau 1. O algoritmo X-NB foi o que se

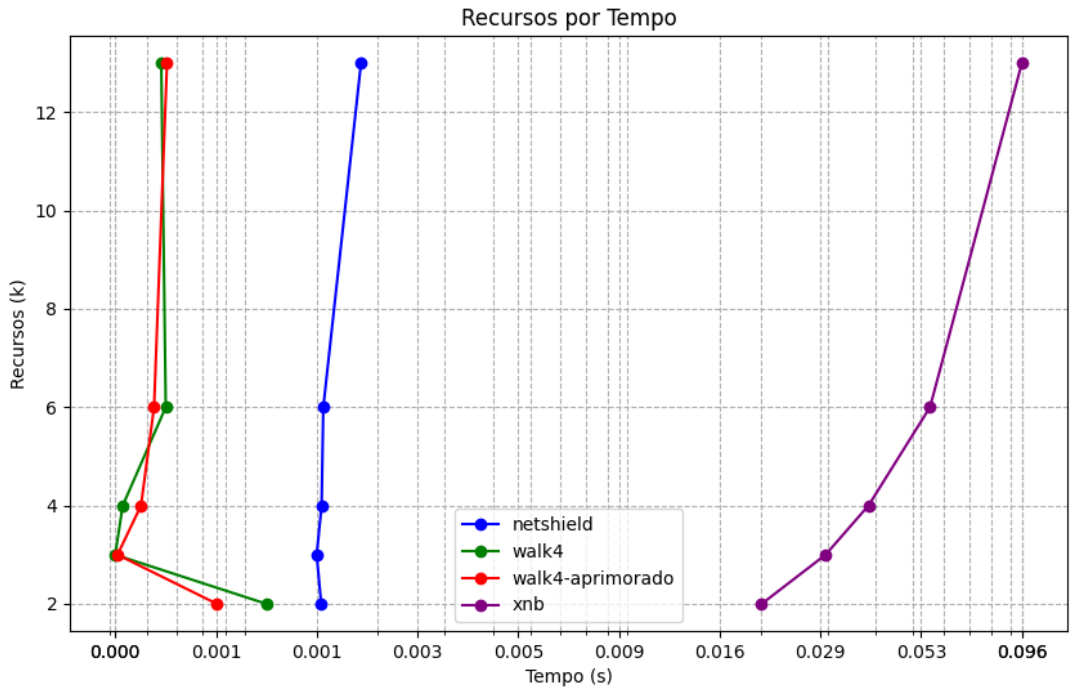


Figura 5.5: Gráfico de recursos por tempo para o grafo gen_graph. 50 vértices e 100 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

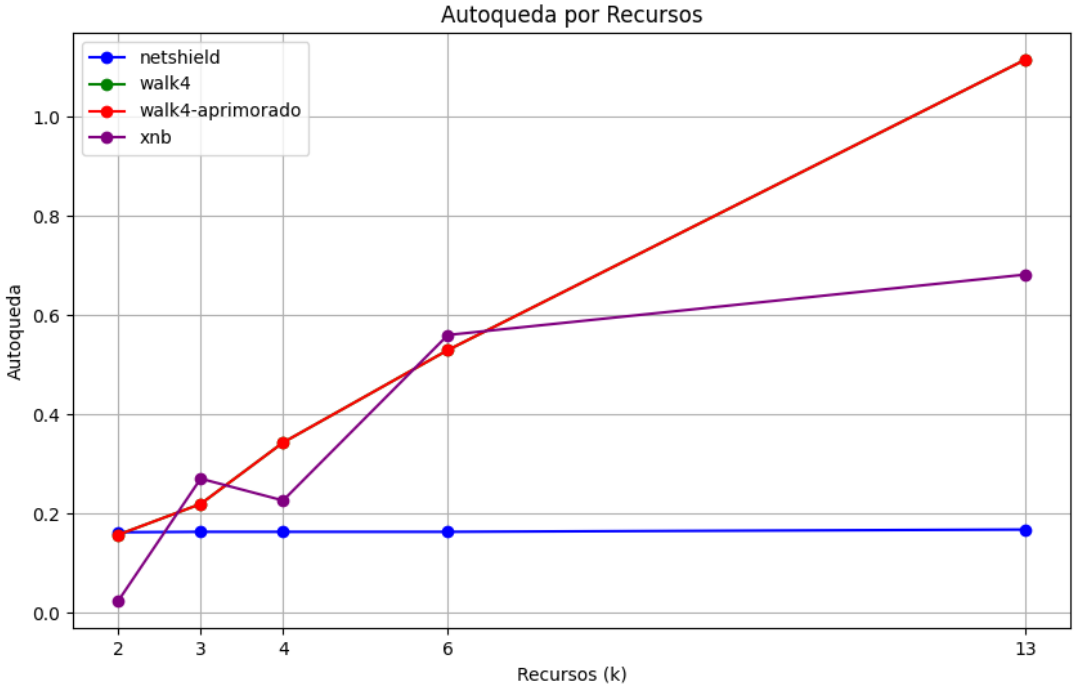


Figura 5.6: Gráfico de autoqueda por número de recursos para o grafo árvore gen_tree, 50 vértices e 100 arestas.

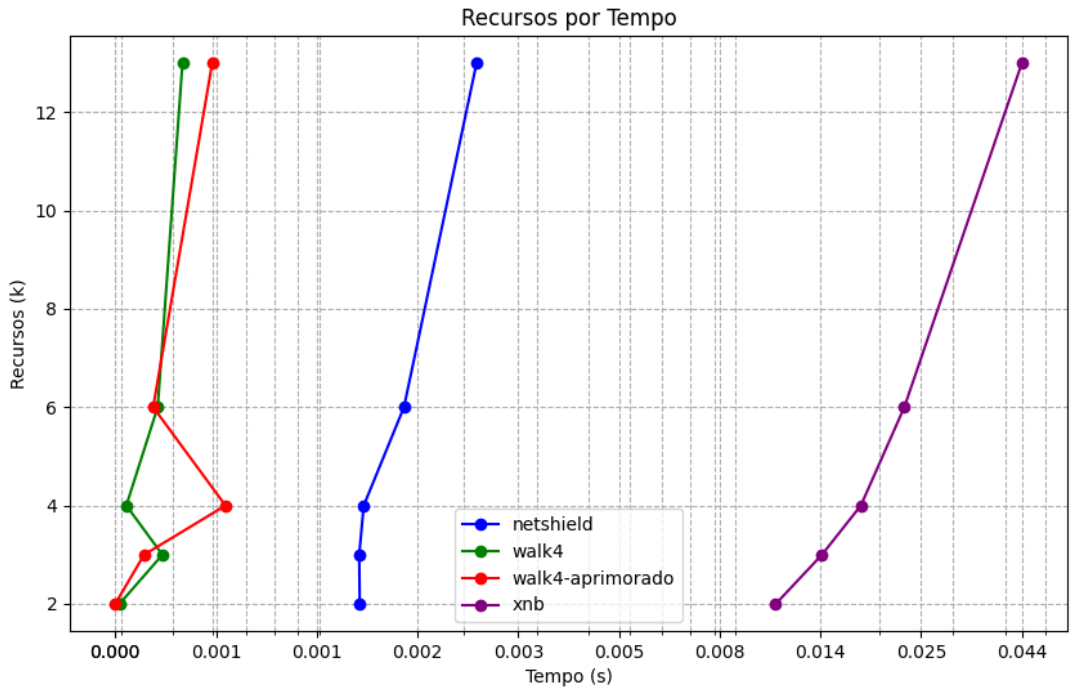


Figura 5.7: Gráfico de recursos por tempo para o grafo árvore gen_tree. 50 vértices e 100 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

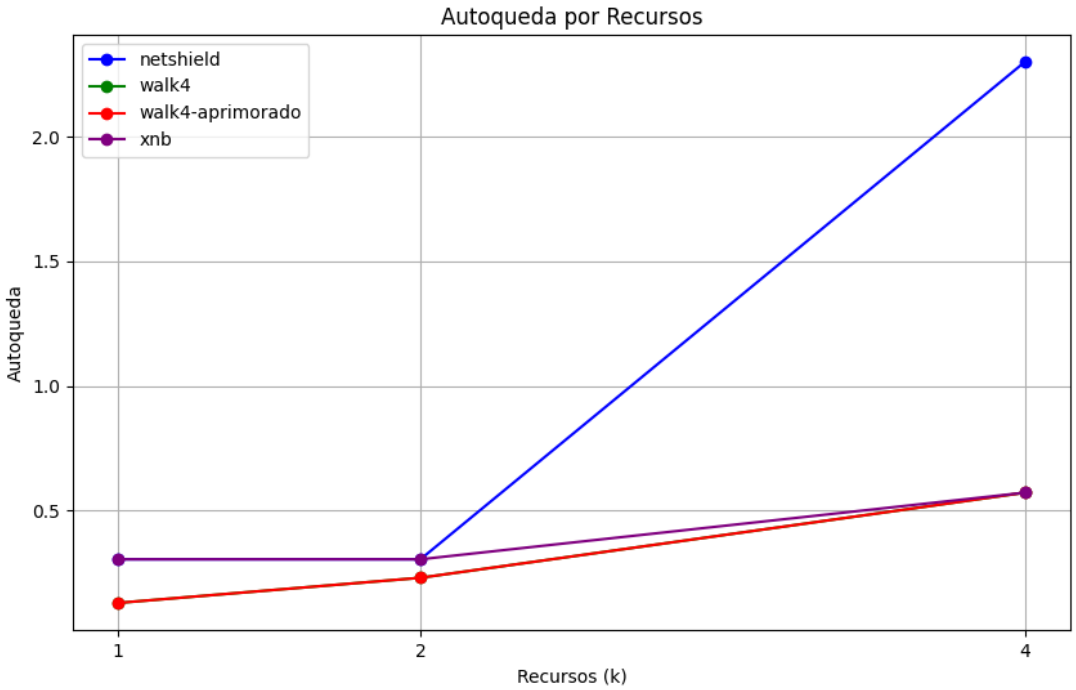


Figura 5.8: Gráfico de autoqueda por número de recursos para o grafo netshield. 16 vértices e 15 arestas.

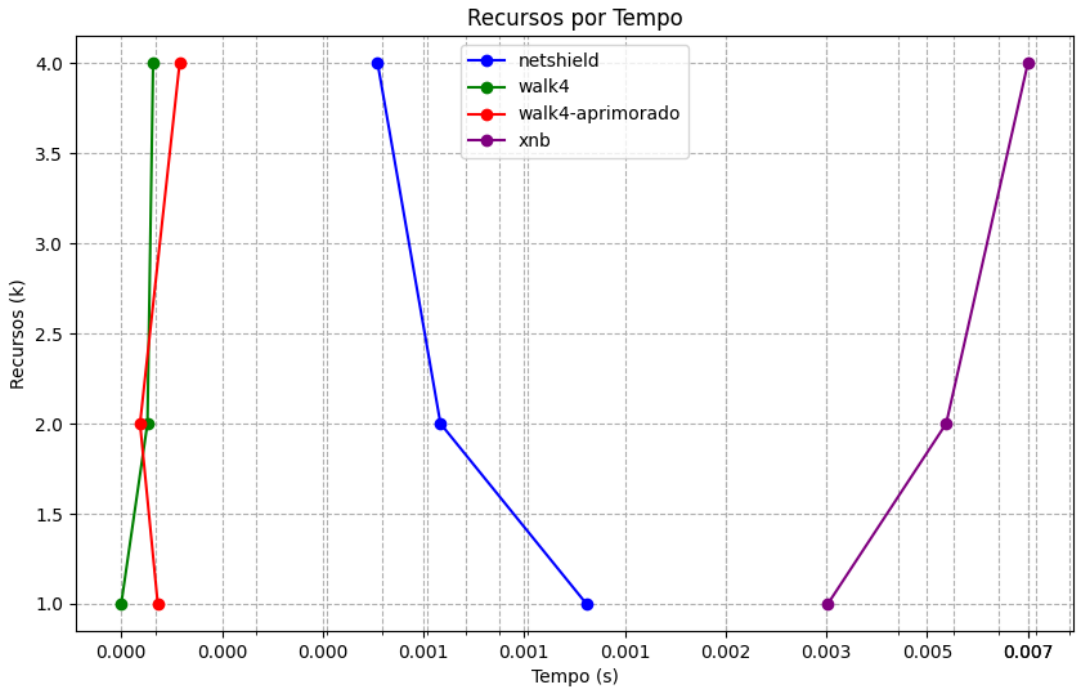


Figura 5.9: Gráfico de recursos por tempo para o grafo netshield. 16 vértices e 15 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

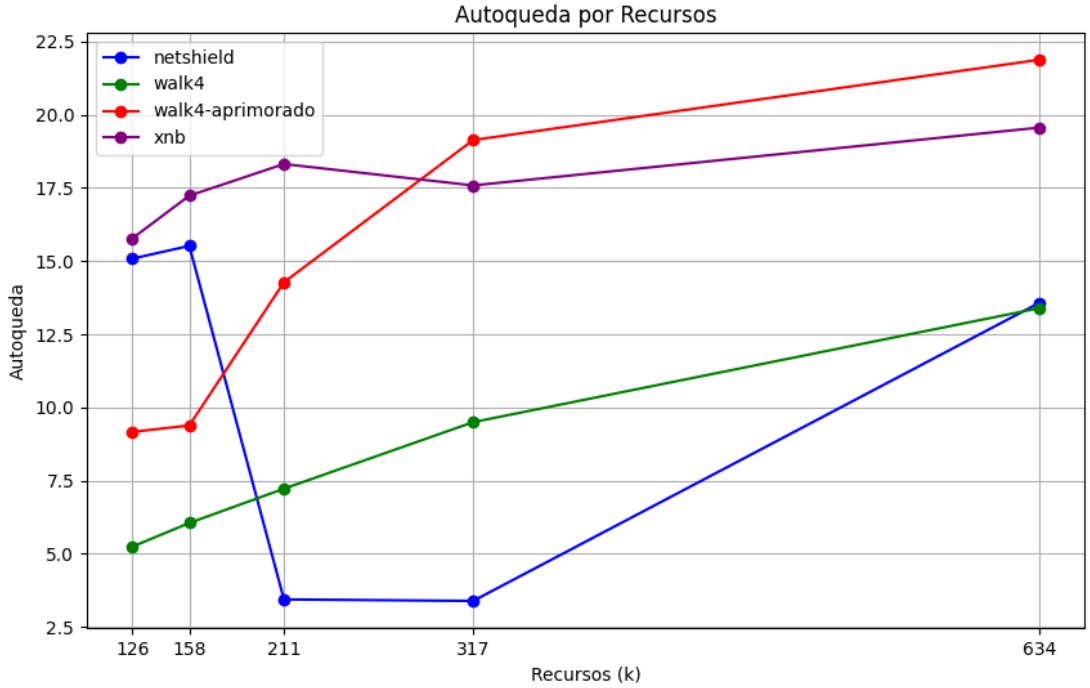


Figura 5.10: Gráfico de autoqueda por número de recursos para o grafo seguidores. 2536 vértices e 3921 arestas.

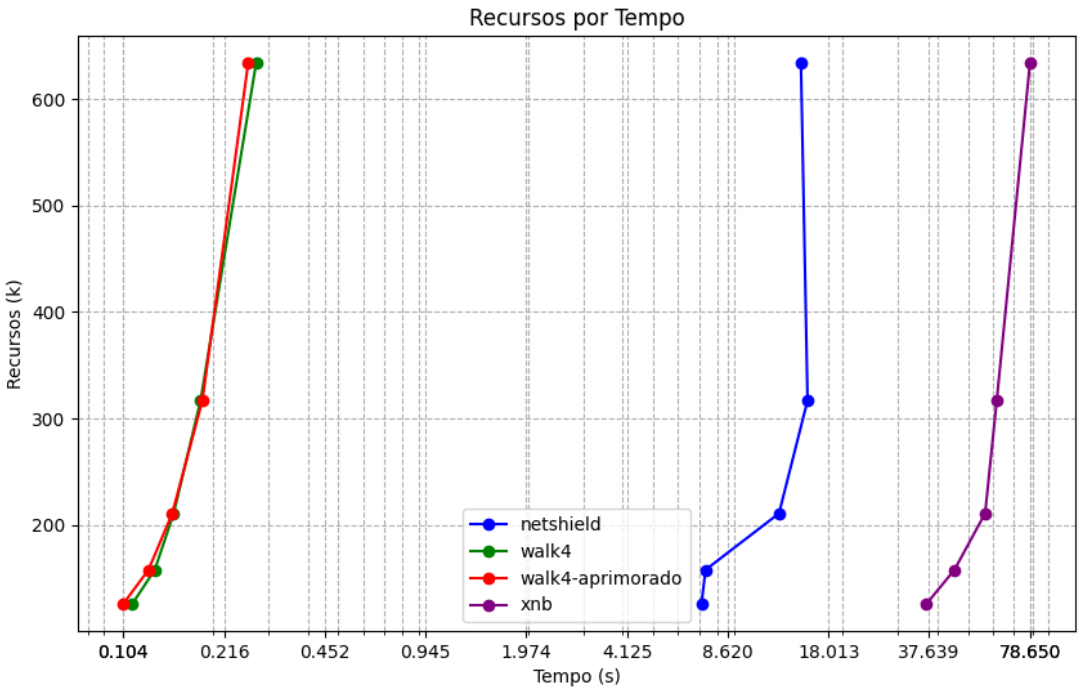


Figura 5.11: Gráfico de recursos por tempo para o seguidores. 2536 vértices e 3921 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

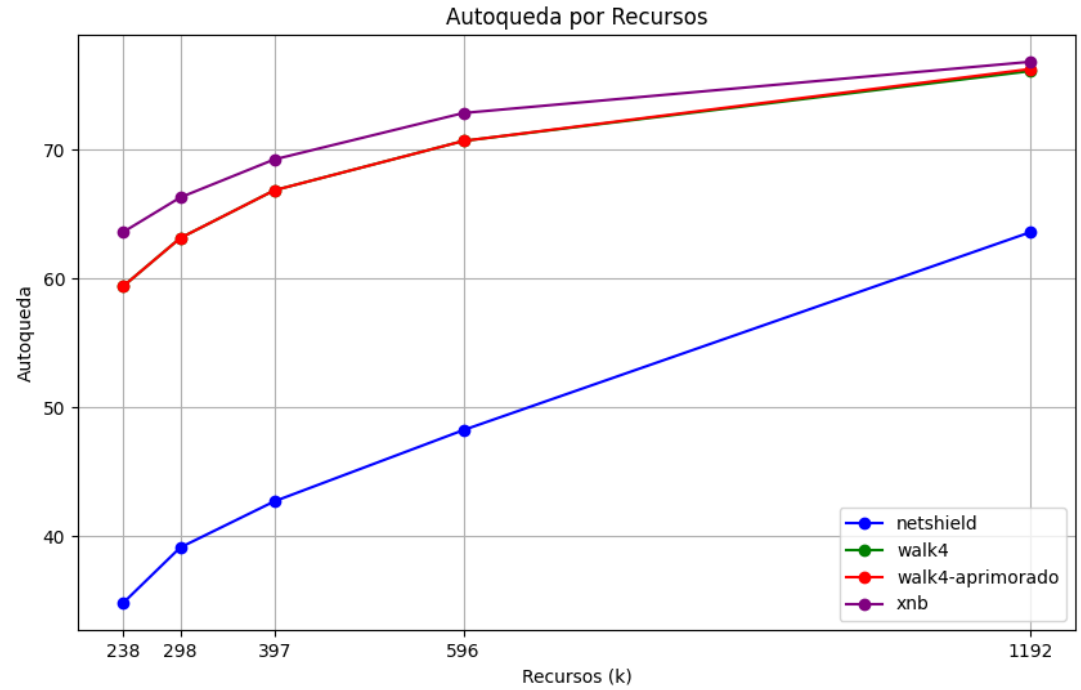


Figura 5.12: Gráfico de autoqueda por número de recursos para o grafo web-spam. 4767 vértices e 37375 arestas.

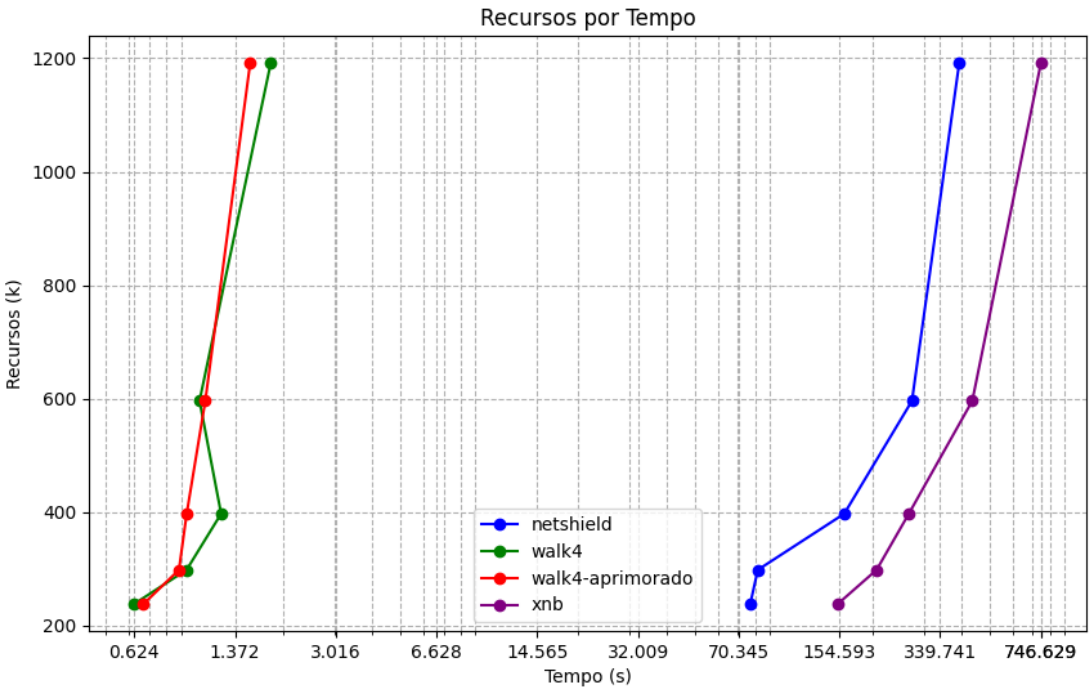


Figura 5.13: Gráfico de recursos por tempo para o web-spam. 4767 vértices e 37375 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

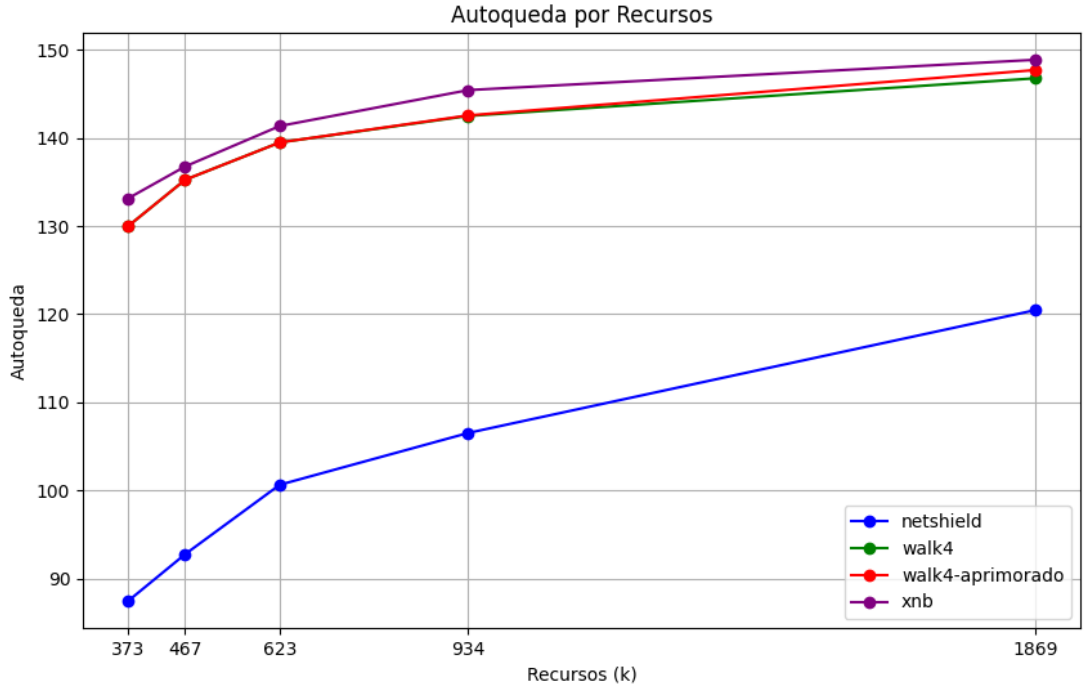


Figura 5.14: Gráfico de autoqueda por número de recursos para o grafo tech-WHOIS. 7476 vértices e 56943 arestas.

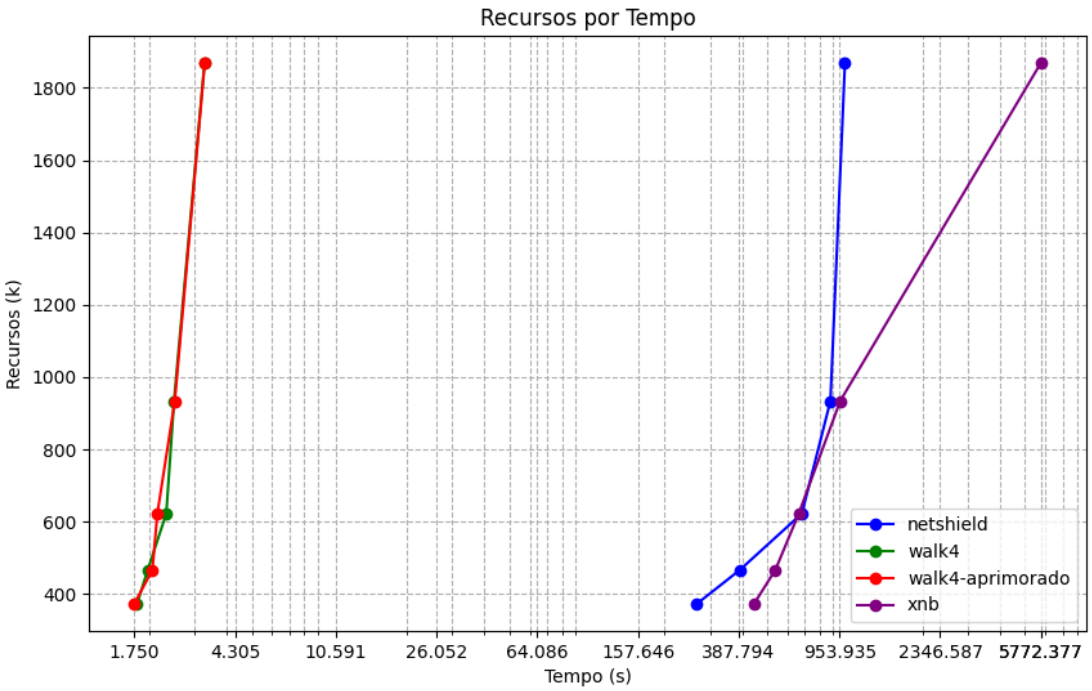


Figura 5.15: Gráfico de recursos por tempo para o tech-WHOIS. 4767 vértices e 37375 arestas. Pode ser lido como "quanto tempo cada algoritmo levou para produzir uma autoqueda".

saiu melhor nos grafos verdadeiramente grandes, mas perceba que seu tempo de execução foi extremamente elevado em relação aos algoritmos de passeios fechados e, até mesmo, em relação ao NetShield. Em todos os casos, menos no caso do grafo netshield, o algoritmo NetShield se saiu pior que os demais. Isso corrobora com a literatura, já que todos os algoritmos comparados aqui foram retirados de artigos que já se comparavam com o próprio NetShield original.

Concluindo, em uma análise de imunização pura e simples, o algoritmo X-NB foi o melhor para grafos grandes. Porém, seu tempo de execução elevado pode ser desincentivador para alguns usos, inclusive. Em seguida, as melhorias simples do algoritmo Walk-4 provaram-se válidas, uma vez que, em nenhum caso, o algoritmo foi expressivamente pior que seu algoritmo de origem; sendo, inclusive, superior a ele em alguns cenários. Por fim, o algoritmo NetShield se mostrou obsoleto para grafos grandes e até mesmo para grafos pequenos. Havia uma expectativa com relação a sua capacidade de imunizar árvores, devido ao sucesso em imunizar corretamente o grafo netshield (figura 5.8), mas ela não foi atendida, como vemos pelo gráfico 5.7.

Capítulo 6

Conclusão

6.1 Discussão

A Teoria Espectral dos Grafos tem se mostrado uma ferramenta indispensável para a análise de redes complexas, oferecendo uma perspectiva algébrica capaz de quantificar e reduzir a vulnerabilidade dessas redes. O poder dessa abordagem reside na sua capacidade de traduzir propriedades estruturais de um grafo em informações quantitativas, como autovalores e autovetores, que podem ser diretamente associadas à resiliência da rede frente a ataques ou à propagação de vírus. A partir desses cálculos, obtemos um controle crucial sobre as intervenções que podemos aplicar para diminuir a vulnerabilidade da rede.

Entretanto, o uso dessa abordagem não é trivial e demanda uma combinação de intuição, criatividade e rigor matemático. A vasta produção científica na área reflete a complexidade do problema e a diversidade de soluções propostas, cada uma trazendo vantagens e limitações em termos de tempo de execução, precisão e aplicabilidade. Através dos algoritmos discutidos e comparados neste estudo, o leitor é introduzido às técnicas atuais utilizadas para abordar o problema da imunização de redes. Espera-se que esse estudo ofereça não apenas uma visão geral, mas também inspire novas abordagens para a resolução desse problema.

Outro aspecto relevante é a consideração de como os recursos são distribuídos e quem tem acesso a eles. Além das questões técnicas, as implicações sociais e éticas das estratégias de imunização de redes devem ser cuidadosamente avaliadas. A imunização, seja no contexto biológico, digital ou social, envolve decisões que podem impactar significativamente grandes populações ou sistemas, o que torna necessário ponderar sobre a equidade e a justiça social. A imunização como forma de identificação de indivíduos mais vulneráveis à propaganda, analogamente ao problema de Influência em Redes, é um exemplo de como essas implicações podem se manifestar.

6.2 Trabalhos futuros

Um aspecto crítico a ser destacado é que nossa análise esteve focada nas técnicas baseadas na Álgebra Linear, sem abordar metodologias combinatórias ou análises específicas de famílias de grafos. Isso abre um vasto campo de pesquisa para o futuro. Por exemplo, a identificação de grafos pertencentes a determinadas famílias conhecidas pode permitir a aplicação de estratégias de imunização otimizadas. No caso de grafos bipartidos, a imunização eficiente pode ser alcançada protegendo-se apenas um dos conjuntos do grafo. Para grafos completos, por outro lado, sabe-se que a imunização completa requer $n - 1$ recursos. A exploração de como as

estratégias algébricas se comportam em diferentes famílias de grafos, e como essas estratégias podem ser combinadas com abordagens combinatórias, é uma área promissora para investigação futura.

Além disso, este estudo focou-se em modelos epidemiológicos de cascata, como SIS e SIR. No entanto, o campo da imunização de redes é vasto e multifacetado. Futuras pesquisas poderiam explorar a imunização de redes que já estejam parcialmente infectadas, considerar a aplicação de modelos epidemiológicos mais complexos, como os que envolvem grafos direcionados ou ponderados, ou mesmo investigar as interações entre diferentes tipos de redes e agentes patogênicos. A introdução de novos modelos epidemiológicos ou a adaptação dos modelos existentes para cenários mais realistas poderia trazer ideias valiosas para a prática da imunização de redes.

Outro fator importante para a aplicabilidade real dessas soluções é a consideração de outras abordagens dentro da Teoria de Grafos. Por exemplo, grafos ponderados poderiam ser usados para representar a probabilidade de uma pessoa infectar outra em uma rede. Grafos direcionados podem ser utilizados para modelar estruturas de dispositivos, como servidores em nuvem, que recebem e processam dados, mas não necessariamente espelham o que recebem. Outra linha de estudo interessante seria investigar se outras matrizes, dentro da Teoria Espectral dos Grafos, podem ser úteis para a análise de vulnerabilidade, além das matrizes já conhecidas e utilizadas.

Em resumo, este estudo oferece uma contribuição significativa para o campo da Teoria Espectral dos Grafos aplicada à imunização de redes, mas também aponta para um horizonte de pesquisa vasto e ainda em grande parte inexplorado. A combinação de diferentes abordagens, a exploração de novas famílias de grafos, a reformulação de modelos epidemiológicos e a consideração das implicações éticas são apenas algumas das direções que podem ser seguidas para avançar ainda mais neste campo de estudo.

Referências Bibliográficas

Ahmad, M., Tariq, J., Shabbir, M. e Khan, I. (2017). Spectral methods for immunization of large networks. *Australasian Journal of Information Systems*, 21.

Backstrom, L., Huttenlocher, D., Kleinberg, J. e Lan, X. (2006). Group formation in large social networks: membership, growth, and evolution. Em *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, página 44–54, New York, NY, USA. Association for Computing Machinery.

Boyd, R. e Richerson, P. (2009). Culture and the evolution of human cooperation. *Philos Trans R Soc Lond B Biol Sci*, 364(1533):3281–8.

Briesemeister, L., Lincoln, P. e Porras, P. (2003). WORM '03, New York, NY, USA. Association for Computing Machinery.

Butler, S. (2022). Steve butler - a short course in spectral graph theory. `https://youtu.be/jB6uVlQC9M4`. Acessado em 02/05/2024.

C3SL (2024). Mecred. `https://mecred.mec.gov.br`. Acessado em 28/07/2024.

Chakrabarti, D., Wang, Y., Wang, C., Leskovec, J. e Faloutsos, C. (2008). Epidemic thresholds in real networks. *ACM Transactions on Information and System Security*, 10(4):1–26.

Chen, C., Tong, H., Prakash, B. A., Tsourakakis, C. E., Eliassi-Rad, T., Faloutsos, C. e Chau, D. H. (2016). Node immunization on large graphs: Theory and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):113–126.

Ganesh, A., Massoulié, L. e Towsley, D. (2005). The effect of network topology on the spread of epidemics. *Proceedings - IEEE INFOCOM*, 2:1455–1466.

Golub, G. H. e Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press, third edition.

Graphviz (2024). Dot language. `https://graphviz.org/doc/info/lang.html`. Acessado em 28/07/2024.

Heise, R. (2024). Repositório remoto com código dos algoritmos. `https://github.com/richardheise/TCC/tree/main/3-Codigo`. Acessado em 29/04/2024.

ifsmirnov (2024). Test generator for olympiad. `https://github.com/ifsmirnov/jngen`. Acessado em 28/07/2024.

Keeling, M. e Eames, K. (2005). Networks and epidemic models. *J R Soc Interface*, 2(4):295–307.

Kempe, D., Kleinberg, J. e Tardos, E. (2003). Maximizing the spread of influence through a social network. Em *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, página 137–146, New York, NY, USA. Association for Computing Machinery.

Liao, H., Mariani, M. S., Medo, M., Zhang, Y.-C. e Zhou, M.-Y. (2017). Ranking in evolving complex networks. *Physics Reports*, 689:1–54. Ranking in evolving complex networks.

Martin, T., Zhang, X. e Newman, M. E. J. (2014). Localization and centrality in networks. *Phys. Rev. E*, 90:052808.

Musawi, A., Roy, S. e Ghosh, P. (2023). Examining indicators of complex network vulnerability across diverse attack scenarios. *Scientific Reports*, 13:18208.

Nemhauser, G. L., Wolsey, L. A. e Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294.

Page, L., Brin, S., Motwani, R. e Winograd, T. (1999). The pagerank citation ranking : Bringing order to the web. Em *The Web Conference*.

Prakash, B., Chakrabarti, D., Valler, N. et al. (2012). Threshold conditions for arbitrary cascade models on arbitrary networks. *Knowledge and Information Systems*, 33(4):549–575.

Radicchi, F. e Castellano, C. (2016). Leveraging percolation theory to single out influential spreaders in networks. *Phys. Rev. E*, 93:062314.

Rezaei, A. A., Munoz, J., Jalili, M. e Khayyam, H. (2022). Vital node identification in complex networks using a machine learning-based approach.

Rhodes, D. e Johnson, B. (2021). Les miserábles interactions. `https://www.researchgate.net/figure/Les-Miserables-graph_fig2_351656150`.

Rossi, R. A. e Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. Em *AAAI*.

Serre, D. (2002). *Matrices*, volume 216 de *Graduate Texts in Mathematics*. Springer.

Sparrow, E., Wood, J. G., Chadwick, C., Newall, A. T., Torvaldsen, S., Moen, A. e Torelli, G. (2021). Global production capacity of seasonal and pandemic influenza vaccines in 2019. *Vaccine*, 39(3):512–520. Epub 2020 Dec 16.

Spielman, D. A. (2007). Spectral graph theory and its applications. Em *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, páginas 29–38.

Tariq, J., Ahmad, M., Khan, I. e Shabbir, M. (2017). Scalable approximation algorithm for network immunization. Em *Pacific Asia Conference on Information Systems*.

Torres, L., Chan, K. S., Tong, H. e Eliassi-Rad, T. (2020). Node immunization with non-backtracking eigenvalues. *ArXiv*, abs/2002.12309.

West, D. B. (2000). *Introduction to Graph Theory*. Prentice Hall, 2 edition.

West, D. B. (2001). *Introduction to Graph Theory*, volume 2. Prentice Hall, Upper Saddle River.

Zhao, N., Yang, S., Wang, H., Zhou, X., Luo, T. e Wang, J. (2024). A novel method to identify key nodes in complex networks based on degree and neighborhood information. *Applied Sciences*, 14(2).

Zhou, M., Tan, J., Liao, H., Wang, Z. e Mao, R. (2020). Dismantling complex networks based on the principal eigenvalue of the adjacency matrix. *Chaos*, 30(8):083118.