



## PROPOSTA E AVALIAÇÃO DE UM ALGORITMO PARA PROMOVER ATAQUES DE QUEBRA DE PRIVACIDADE EM REDES ANONIMIZADAS

Pamela Tabak

Projeto de Graduação apresentado ao Curso de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro  
Julho de 2017

PROPOSTA E AVALIAÇÃO DE UM ALGORITMO PARA PROMOVER  
ATAQUES DE QUEBRA DE PRIVACIDADE EM REDES ANONIMIZADAS

Pamela Tabak

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE  
COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA  
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE  
DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE  
ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

---

Prof. Daniel Ratton Figueiredo, Dr.

---

Prof. Fernando Gil Vianna Resende Junior, Dr.

---

Prof. Flávio Luis de Mello, Dr.

RIO DE JANEIRO, RJ – BRASIL  
JULHO DE 2017

Tabak, Pamela

Proposta e Avaliação de um Algoritmo para Promover Ataques de Quebra de Privacidade em Redes Anonimizadas/Pamela Tabak. – Rio de Janeiro: UFRJ/POLI – COPPE, 2017.

X, 57 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Computação e Informação, 2017.

Referências Bibliográficas: p. 56 – 57.

1. Grafos. 2. Redes Sociais. 3. Anonimização.
4. Privacidade em Mineração de Dados. 5. Redes Complexas. 6. Ataques em Grafos. I. Figueiredo, Daniel Ratton. II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Computação e Informação. III. Título.

# Agradecimentos

Agradeço ao meu pai, que sempre me incentivou e me ensinou que dedicação e esforço trazem bons resultados. À minha mãe, que desde pequena me mostrou o que era ter força e determinação e sempre acreditou que eu seria capaz de ser boa naquilo que eu gostasse. Aos dois, agradeço pelo amor incondicional e todo apoio fornecido, além de me mostrarem o caminho para o mundo da engenharia.

Agradeço a todos os meus amigos, pessoas com quem eu puder contar e confiar em todos os momentos e me fizeram ter lembranças e histórias maravilhosas para contar. Em especial, ao meu melhor amigo e namorado, Eric Reis, que me apoiou ao longo deste projeto, assim como durante toda a faculdade.

À Universidade Federal do Rio De Janeiro sou grata por ter passado os últimos anos aqui. Muito antes de saber o curso que iria fazer, já tinha escolhido a universidade que desejava estudar. Aos professores do curso, agradeço por todas as aulas lecionadas, dentro e fora de sala, que me deram as ferramentas necessárias para me tornar engenheira.

Por fim, agradeço ao professor Daniel Figueiredo, por todos os ensinamentos passados ao longo da faculdade e pela disposição e motivação em orientar este projeto.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

## **PROPOSTA E AVALIAÇÃO DE UM ALGORITMO PARA PROMOVER ATAQUES DE QUEBRA DE PRIVACIDADE EM REDES ANONIMIZADAS**

**Pamela Tabak**

Julho/2017

Orientador: Daniel Ratton Figueiredo

Curso: Engenharia de Computação e Informação

Os traços digitais da interação social humana são facilmente encontrados ao redor da *Web*. Sites com relacionamentos sociais entre seus usuários têm se tornado cada vez mais comuns e são naturalmente adotados em larga escala. Muitas dessas redes possuem dados sensíveis e, conseqüentemente, têm caráter privado, de forma a proteger a privacidade de seus usuários. Entretanto, frequentemente tais dados são tornados públicos depois de passarem por um processo de remoção da identidade dos usuários, também conhecido por anonimização, de modo a permitir a realização de estudos empíricos a partir dessas informações. Nesse contexto, ataques de quebra de privacidade surgem para revelar relações entre usuários de uma determinada rede. Este trabalho propõe e avalia um método para realizar um ataque contra a privacidade dos usuários de uma rede. O método insere nós artificiais na rede e estabelece um padrão de conexões incidentes a eles. Uma vez que a rede tenha sido anonimizada, o objetivo é identificar tais vértices, quebrando a privacidade de outros vértices da rede, conseqüentemente. Iremos avaliar a eficiência do método, utilizando modelos matemáticos de redes e redes reais.

**Palavras-Chave:** Grafos, Redes Sociais, Anonimização, Privacidade em Mineração de Dados, Redes Complexas, Ataques em Grafos.

Abstract of the Undergraduate Project presented to Poli/COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

## **PROPOSAL AND EVALUATION OF AN ALGORITHM TO PROMOTE PRIVACY BREACH IN ANONYMIZED NETWORKS**

**Pamela Tabak**

July/2017

Advisor: Daniel Ratton Figueiredo

Course: Computer and Information Engineering

The digital traces of human social interactions are easily found all over the Web. Websites with social relationships among their users are increasingly common and are naturally adopted on a large scale. Many of those networks have sensitive data and, therefore, are private, in order to protect the privacy of their users. However, such data are often made public after passing through a process that removes the identity of the users, also known as anonymization, in order to allow the realization of empirical studies based on this information. In this context, privacy breach attacks arise in order to reveal relationships between users from a specific network. This work proposes and evaluates a method to promote an attack against the privacy of the users inside a network. The method inserts artificial nodes in the network and establishes a connection pattern. Once the network has been anonymised, the goal is to identify those nodes, breaking the privacy of other nodes inside the network, consequently. We will evaluate the efficiency of the method, using real networks and mathematical models of networks.

**Keywords:** Graphs, Social Networks, Anonimization, Privacy in Data Mining, Complex Networks, Graph Attacks.

# Sumário

<b>Lista de Figuras</b>	<b>ix</b>
<b>Lista de Tabelas</b>	<b>x</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Tipos de ataque . . . . .	2
1.1.1 Ativo . . . . .	2
1.1.2 Passivo . . . . .	3
1.1.3 Híbrido . . . . .	3
1.2 Motivação . . . . .	3
1.3 Objetivo . . . . .	4
1.4 A técnica . . . . .	5
1.5 Estruturação do documento . . . . .	6
<b>2 Conceitos e trabalhos relacionados</b>	<b>8</b>
2.1 Conceitos . . . . .	8
2.1.1 Automorfismo . . . . .	8
2.1.2 Isomorfismo . . . . .	9
2.2 Trabalhos relacionados . . . . .	9
2.2.1 <i>Anonymized social networks, hidden patterns, and structural steganography</i> . . . . .	9
2.2.2 <i>Resisting structural re-identification in anonymized social networks</i> . . . . .	12
<b>3 Algoritmo proposto</b>	<b>14</b>
3.1 Explicação detalhada . . . . .	14
3.1.1 Definições matemáticas . . . . .	14
3.1.2 Criação dos nós atacantes . . . . .	16
3.1.3 Anonimização do grafo . . . . .	26
3.1.4 Recuperação do subgrafo . . . . .	29
3.2 Implementação . . . . .	35

<b>4</b>	<b>Avaliação</b>	<b>36</b>
4.1	Metodologia da avaliação . . . . .	36
4.2	Resultados . . . . .	37
4.2.1	Grafos aleatórios . . . . .	37
4.2.2	Redes reais . . . . .	41
<b>5</b>	<b>Detecção do ataque</b>	<b>48</b>
5.1	Detecção de comunidade . . . . .	48
5.1.1	Método de Louvain . . . . .	49
5.1.2	Resultados . . . . .	50
<b>6</b>	<b>Conclusão e trabalhos futuros</b>	<b>53</b>
6.1	Conclusão . . . . .	53
6.2	Trabalhos futuros . . . . .	54
6.2.1	Maior abrangência . . . . .	54
6.2.2	Comparações . . . . .	54
6.2.3	Grafos direcionados . . . . .	54
6.2.4	Remoção de arestas . . . . .	54
6.2.5	Detecção do ataque . . . . .	55
	<b>Referências Bibliográficas</b>	<b>56</b>



# Lista de Figuras

1.1	<i>Exemplo de um grafo anonimizado após sofrer o ataque . . . . .</i>	6
2.1	<i>Automorfismo em grafo . . . . .</i>	8
2.2	<i>Isomorfismo em grafos . . . . .</i>	9
3.1	<i>Exemplo do subgrafo de atacantes após adição inicial de arestas . . .</i>	18
3.2	<i>Exemplo do subgrafo de atacantes após adição das arestas aleatórias entre os atacantes . . . . .</i>	18
3.3	<i>Fluxograma explicando as ligações entre nós atacantes e nós vítimas .</i>	20
3.4	<i>Fluxograma explicando a geração de grau externo máximo . . . . .</i>	21
3.5	<i>Fluxograma explicando as ligações entre nós atacantes e nós não vítimas</i>	23
3.6	<i>Fluxograma explicando a verificação de automorfismos . . . . .</i>	25
3.7	<i>Grafo para exemplificar componente de anonimização . . . . .</i>	27
3.8	<i>Grafo anonimizado pelo processo de anonimização a partir do índice de cada nó . . . . .</i>	28
3.9	<i>Grafo anonimizado pelo processo de anonimização a partir da permutação entre os identificadores . . . . .</i>	29
3.10	<i>Fluxograma explicando a busca pelo nó atacante inicial . . . . .</i>	30
3.11	<i>Fluxograma explicando a busca pelos demais nós atacantes . . . . .</i>	31
3.12	<i>Árvore exemplificando a recuperação dos atacantes no grafo anonimizado . . . . .</i>	32
3.13	<i>Árvore representando o sucesso na recuperação dos atacantes no grafo anonimizado . . . . .</i>	33
3.14	<i>Exemplo de árvore representando falha causada por isomorfismo . . .</i>	33
3.15	<i>Exemplo de árvore representando falha causada por automorfismo . .</i>	34

# Lista de Tabelas

3.1	Tabela representando vetor de marcação de atacantes a cada instante, com 4 atacantes . . . . .	20
3.2	Tabela representando resultados do processo de anonimização a partir do índice de cada nó . . . . .	27
3.3	Tabela representando resultados do processo de anonimização a partir da permutação entre os identificadores . . . . .	28
4.1	Tabela representando desempenho do algoritmo no modelo Erdős-Rényi	38
4.2	Tabela representando desempenho do algoritmo no modelo Barabási-Albert . . . . .	40
4.3	Tabela representando desempenho do algoritmo em uma rede do Facebook . . . . .	42
4.4	Tabela representando desempenho do algoritmo na rede de votação da Wikipédia . . . . .	44
4.5	Tabela representando desempenho do algoritmo na rede de compartilhamento de arquivo Gnutella P2P . . . . .	46
5.1	Tabela representando informações sobre as comunidades na rede do Facebook . . . . .	51
5.2	Tabela representando informações sobre as comunidades na rede Gnutella P2P . . . . .	51
5.3	Tabela representando informações sobre as comunidades na rede do modelo Erdős-Rényi . . . . .	52

# Capítulo 1

## Introdução

Os traços digitais da interação social humana são facilmente encontrados ao redor da *Web*. Sites com relacionamentos sociais entre seus usuários têm se tornado cada vez mais comuns e são naturalmente adotados em larga escala, para diversos fins. Consequentemente, essas fontes se tornaram, nos últimos anos, dados importantes para grandes estudos envolvendo redes sociais.

Em muitas dessas fontes de dados o relacionamento entre os usuários é público, isto é, a informação sobre dois usuários da rede estarem conectados pode ser obtida por qualquer outro usuário. Esse seria o caso do Facebook, por exemplo. Entretanto, existem muitas redes sociais em que essas relações não são disponibilizadas, de modo que cada usuário conheça apenas as relações que ele possui, desconhecendo a existência de relações entre dois outros usuários da rede. Um exemplo desse tipo seria a rede de troca de e-mail: cada usuário sabe para quem enviou e de quem recebeu e-mails, porém ele não é capaz de saber se um usuário qualquer enviou ou recebeu e-mail de outro usuário. Nestas redes cada usuário é capaz de determinar o valor do seu próprio grau, porém a única informação que eles podem assumir, relativa ao grau de seus vizinhos, é que vale pelo menos um, uma vez que este está conectado ao usuário.

Assim como na rede de e-mail citada, é comum que muitas das redes tenham dados sensíveis e, consequentemente, tenham caráter privado, de forma a proteger a privacidade de seus usuários. Entretanto, frequentemente empresas divulgam a sua rede anonimizada, isto é, a estrutura da rede é mantida porém sem a identificação de cada usuário, de modo a permitir que estudos empíricos sejam feitos. Um caso assim foi o da Netflix, descrito em [1], que criou uma competição online para o desenvolvimento de um sistema de recomendação que fosse superior ao que era até então utilizado pela empresa. Para a criação desse sistema não era necessário ser capaz de identificar os nós da rede, compostos por usuários e filmes, de modo que a empresa tenha anonimizado os dados antes de fazer a divulgação da rede. Dessa forma, milhares de pessoas ao redor do mundo puderam dar ideias para a criação

desse sistema e aprimorá-lo sem que conhecessem as relações reais existentes entre os usuários e os filmes.

Muitas vezes, antes de uma empresa divulgar a rede anonimizada, ela anuncia que fará essa liberação de dados, por diversos motivos. Sendo assim, é durante esse período que um ataque pode ocorrer: sabendo que um grafo anonimizado que representa uma rede será disponibilizado, o ataque irá inserir nós artificiais na rede e um padrão de conexões incidentes a estes nós, antes dos dados serem anonimizados e tornados públicos. Assim, uma vez que a rede tenha sido anonimizada e disponibilizada, o objetivo será procurar por esses nós a partir da estrutura da rede, uma vez que o promotor do ataque possui informações estruturais sobre os nós criados por ele. Ao identificar estes nós, a identificação de nós originais da rede será possível, revelando relações entre estes usuários, que até então eram privadas, de modo a quebrar suas privacidades. Um outro possível cenário é adicionar os nós e arestas maliciosamente na rede, na esperança que os dados sejam divulgados futuramente, acidentalmente ou não.

## 1.1 Tipos de ataque

Nessa conjectura, existem algumas abordagens de ataques proeminentes [2]: ativo, passivo e híbrido. Em todos os ataques citados, o promotor do ataque precisa definir um conjunto de nós da rede a ser atacado, isto é, a ter sua privacidade violada, revelando suas relações com outros usuários da rede.

### 1.1.1 Ativo

Ataques ativos são algoritmos que adicionam novos usuários à rede, chamados de nós atacantes, além de adicionar novas relações, tanto entre esses nós quanto entre os nós atacantes e os nós atacados, ou vítimas. Nesse contexto, nós vítimas são os usuários da rede que o promotor do ataque escolheu para violar a privacidade e relações entre nós são as arestas de um grafo.

Como o promotor do ataque insere nós e arestas no grafo propositalmente, ele passa a ter alguma informação sobre a estrutura da rede, que até então era completamente desconhecida para ele. Informações estruturais sobre os vértices que ele criou que não dependam de outras métricas, como o grau, passam a ser dados conhecidos, uma vez que o atacante criou estes nós e as relações entre eles. A ideia é, portanto, a partir dessas informações, encontrar os nós atacantes adicionados no grafo anonimizado, de modo a identificar os nós vítimas a partir da maneira como as relações foram criadas entre estes e os atacantes.

Neste ataque, qualquer usuário da rede pode ser escolhido para ser atacado.

### 1.1.2 Passivo

Neste ataque, os promotores do ataque são alguns dos próprios usuários do sistema a ser atacado. Eles não criam novos usuários ou novas arestas, usando a informação estrutural que eles já possuem da rede, deles mesmos, para encontrá-los no grafo anonimizado. A partir disto, eles tentam identificar os seus vizinhos, de modo a encontrar arestas existentes entre estes, revelando relações entre usuários.

No ataque passivo, por sua vez, apenas usuários relacionados ao grupo de usuários atacantes podem ser atacados, limitando a informação que será extraída da rede. Entretanto, este ataque não altera a estrutura da rede, de modo que ele seja mais dificilmente detectado como um possível ataque.

### 1.1.3 Híbrido

Existem prós e contras em relação aos ataques passivo e ativo descritos. Os ataques ativos têm efeitos mais potentes, uma vez que eles não precisam depender na chance de um usuário (ou um conjunto de usuários) encontrar-se de forma única no grafo anonimizado, além do atacante poder escolher qualquer usuário da rede para violar a privacidade. Por outro lado, enquanto o ataque passivo é apenas capaz de comprometer os usuários que estão conectados ao grupo de atacantes, ele possui uma característica impressionante: o atacante é simplesmente um usuário do sistema que vê a rede anonimizada, não alterando a estrutura da rede, de modo que não exista informação indesejada que possa ser detectada. O ataque híbrido, também conhecido por ataque semi-passivo, portanto, surgiu para tentar juntar as qualidades dos dois ataques.

Neste caso, os promotores do ataque também são usuários do sistema. Eles não criam nenhum novo usuário, porém criam algumas novas arestas entre eles e os usuários escolhidos como vítimas antes da rede ser divulgada, de forma a poder encontrar relações entre quaisquer usuários da rede e não apenas aqueles aos quais eles já estavam previamente conectados.

## 1.2 Motivação

É inegável afirmar que as redes sociais se tornaram parte do cotidiano de grande parte dos usuários da Internet. O Facebook, um dos maiores exemplos do início do século XXI, chegou a 1.6 bilhão de usuários em 2016, cerca de metade da população mundial com acesso à Internet contabilizada neste mesmo ano.

Neste contexto, é possível perceber que com o aumento de usuários da rede, mais fácil se torna a adesão de novos usuários. Primeiro porque a rede social em questão se torna cada vez mais popular e atinge mais pessoas conforme o número de

usuários aumenta. Segundo, as pessoas tendem a se sentir mais seguras em relação à um serviço quando várias outras conhecidas estão o utilizando. A dúvida é se os dados disponibilizados estão realmente seguros nestas redes.

A privacidade em redes, e na Web em geral, continua sendo um assunto importante e muito estudado. De tempos em tempos é possível ver no noticiário grandes empresas que tiveram parte de seus dados privados divulgados, o que acabou com a privacidade que muitas pessoas acreditavam ter sobre seus dados.

Dessa forma, a criação de ataques contra a privacidade em redes, visando revelar relações entre usuários, se torna um assunto cada vez mais interessante por principalmente dois motivos: é mais fácil se proteger de ataques que são conhecidos, de modo que empresas criem os ataques para testar os níveis de proteção contra os mesmos; informações importantes podem ser extraídas dessas redes. Voltando ao exemplo da rede de e-mail, um possível caso de informação que poderia ser extraída seria a confirmação da troca de mensagens entre duas pessoas, que poderia ser usada em diversos casos jurídicos em que as pessoas envolvidas afirmam que não houve comunicação entre elas.

## 1.3 Objetivo

O objetivo deste trabalho é apresentar o método desenvolvido para identificar nós determinados em redes que não tem informação sobre as relações entre os seus usuários, como a rede de e-mail citada. Isto é, nestas redes, cada usuário conhece apenas as suas relações. A proposta deste trabalho é, portanto, encontrar relações entre os nós escolhidos pelo promotor do ataque, chamados de nós atacados, ou nós vítimas, ao longo deste projeto. Este algoritmo seguirá os princípios de um ataque ativo.

A identificação dos nós escolhidos para serem atacados será feita em grafos não-direcionados e anonimizados, a partir da estrutura da rede. O algoritmo foi elaborado com o intuito de não existir restrição quanto ao número de nós vítimas, isto é, ele visa identificar qualquer quantidade de nós na rede, inclusive todos. Para isso, também é levado em consideração que o número de nós a serem adicionados na rede, por ser um ataque ativo, deverá ser pequeno quando comparado ao tamanho da rede. O cálculo detalhado do número de nós atacantes usado neste projeto é apresentado no capítulo 3.

Além disso, a eficácia do método desenvolvido será analisada, para diferentes conjuntos de nós vítimas. Serão considerados apenas dois possíveis resultados do algoritmo: sucesso, quando todos os nós vítimas foram identificados, e falha, no caso contrário. Para isso, diferentes redes serão analisadas no capítulo 4, como redes aleatórias geradas a partir de modelos matemáticos e redes reais, fazendo um es-

tudo comparativo em relação ao número de nós vítimas. De modo a tornar possível a avaliação do método, isto é, averiguar quantas vezes conseguiu ou não identificar os nós vítimas, este trabalho irá analisar redes não anonimizadas, simulando a anonimização após a adição dos nós atacantes e suas arestas na rede original.

A escolha pelo ataque ativo torna possível atacar qualquer usuário de qualquer rede, sem a necessidade do atacante já estar infiltrado na mesma. A maior desvantagem deste tipo de ataque, segundo [2], é a probabilidade deste tipo de ataque ser detectado, uma vez que ele tenta inserir um subgrafo na rede que se destaque dos demais, para que o próprio promotor do ataque possa posteriormente identificar esse subgrafo a partir de sua estrutura. Este trabalho irá abordar brevemente, no capítulo 5, a possível detecção do ataque. Os grafos não-direcionados, por sua vez, foram escolhidos por possuírem menos informação estrutural (cada nó possui um grau, enquanto no grafo direcionado existem dois - grau de saída e de entrada), sendo um processo mais complexo. Contudo, o algoritmo pode ser facilmente adaptado para grafos direcionados.

## 1.4 A técnica

A técnica desenvolvida terá dois componentes principais: a criação e conexão de nós atacantes e a identificação dos nós vítimas, que será feita a partir da identificação do subgrafo criado no ataque, isto é, o subgrafo composto pelos nós atacantes.

A partir de um grafo não-direcionado e um conjunto de nós a serem atacados, serão criados  $\mathcal{O}(\log n)$  novos nós, atacantes, sendo  $n$  o número de nós da rede original. O conjunto de nós atacantes é representado por  $X = \{a_0, a_1, \dots, a_{x-1}\}$ , sendo  $x$  o número total de atacantes. As arestas entre estes nós serão criadas de maneiras determinísticas e aleatórias, seguindo um padrão de regras definidas pelo ataque, de modo a facilitar a identificação do subgrafo induzido por estes nós no grafo anonimizado.

Além disso, cada nó vítima irá se conectar a um subconjunto único de nós atacantes. Isto é, nenhum par de nós vítimas estará conectado aos exatos mesmos nós atacantes. Assim, se for possível identificar os atacantes no grafo anonimizado, será possível identificar, por definição, todos os nós que estão sendo atacados. Como foi o promotor do ataque quem criou as arestas entre os atacantes e os nós vítimas, ele sabe exatamente qual nó vítima está conectado a cada subconjunto de atacantes.

Por fim, é calculado um grau máximo para cada atacante, maior ou igual ao grau que ele já possui até o momento. Cada atacante irá se conectar a nós não vítimas, até atingir seu grau máximo ou até não ter mais nós a se conectar: quando todos os nós do grafo são vítimas, por exemplo, não existem nós não vítimas para completar o grau dos atacantes. Este processo visa aumentar a aleatoriedade do

subgrafo criado pelo promotor do ataque.

A figura 1.1 representa um exemplo do ataque ocorrendo em um grafo anonimizado de 4 vértices, formado pelos nós azuis e cinzas, que representam nós vítimas (0,1) e não vítimas (2,3), respectivamente. Os nós coloridos por rosa (4,5,6,7), por sua vez, são os nós atacantes, isto é, nós que foram adicionados ao grafo para identificar o subgrafo induzido formado pelos nós azuis.

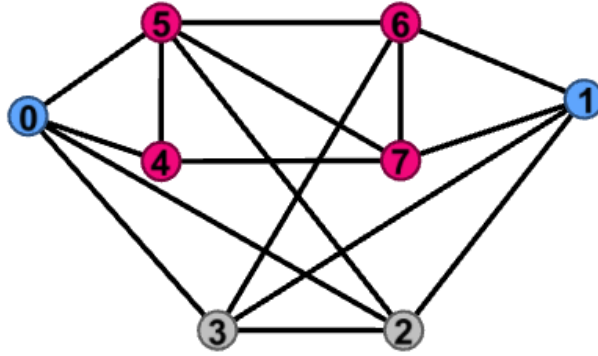


Figura 1.1: Exemplo de um grafo anonimizado após sofrer o ataque.

Para a recuperação do subgrafo criado, o promotor do ataque possui apenas a informação relativa aos nós e arestas que ele criou, uma vez que nas redes estudadas a informação estrutural não é revelada a priori. Este processo irá procurar, a partir dos conhecimentos que possui sobre a estrutura da rede, os nós atacantes em ordem. Isto é, o primeiro passo do processo é procurar pelo nó atacante  $a_0$ , a partir de seu grau e do grau dos seus vizinhos. Dessa forma, o algoritmo irá fazer uma varredura no grafo em busca de nós anonimizados com o mesmo grau do  $a_0$  e com uma sequência de grau dos vizinhos que contenha a sequência de graus dos vizinhos atacantes. Este processo é repetido para todos os nós atacantes, até todos serem identificados ou chegar à conclusão que não é possível fazer a identificação, resultando em falha.

A partir da maneira como os nós vítimas foram relacionados aos nós atacantes, se for possível identificar todos os nós atacantes no grafo anonimizado, então será possível identificar todos os nós vítimas. Como cada nó vítima está conectado a um subconjunto único de nós atacantes e todos os nós atacantes foram identificados na rede, é possível então identificar todos os nós vítimas e todas as relações entre eles.

O algoritmo completo é apresentado detalhadamente no capítulo 3.

## 1.5 Estruturação do documento

No capítulo 2, veremos alguns conceitos que serão utilizados ao longo deste trabalho e dois trabalhos relacionados, isto é, trabalhos que também estudam ataques em grafos anonimizados e detecção dos mesmos. O capítulo 3 detalha o algoritmo



proposto, bem como a tecnologia envolvida no seu desenvolvimento. O capítulo 4 apresenta os resultados de diferentes execuções do método apresentado. O capítulo 5 apresenta uma técnica para detecção do ataque e analisa os resultados da mesma. Por fim, o capítulo 6 apresenta conclusões a respeito do trabalho e possíveis extensões e melhorias em trabalhos futuros.

# Capítulo 2

## Conceitos e trabalhos relacionados

### 2.1 Conceitos

Esta seção explicará dois conceitos de grafos que serão muito utilizados ao longo deste trabalho.

#### 2.1.1 Automorfismo

Segundo [3], um automorfismo de um grafo é uma forma de simetria em que o grafo é mapeado em si, preservando a conectividade vértice-aresta. Formalmente, um automorfismo de um grafo  $G = (V, E)$  é uma permutação  $\sigma$  do conjunto de vértices  $V$ , tal que para qualquer aresta  $e = (u, v)$ ,  $\sigma(e) = (\sigma(u), \sigma(v))$  é também uma aresta. Ou seja, ele é um isomorfismo de grafos de  $G$  para ele mesmo.

A figura 2.1 representa um grafo que contém automorfismo. Apenas a partir da estrutura da rede, nenhum algoritmo é capaz de diferenciar os nós 1 e 3, de modo que eles não são identificados de maneira única.

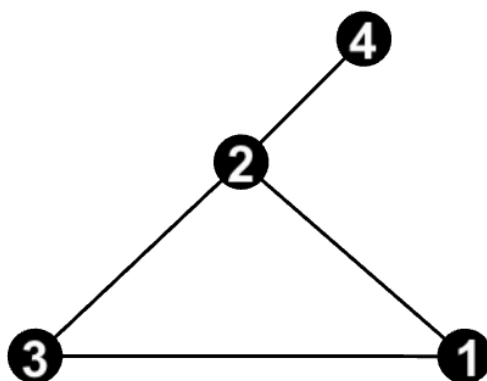


Figura 2.1: Automorfismo em grafo.

### 2.1.2 Isomorfismo

De acordo com [4], um isomorfismo entre dois grafos  $G$  e  $H$  é uma bijeção entre os conjuntos de vértices de  $G$  e  $H$   $f: V(G) \rightarrow V(H)$  de tal forma que quaisquer dois vértices  $u$  e  $v$  de  $G$  são adjacentes em  $G$  se e somente se  $f(u)$  e  $f(v)$  são adjacentes em  $H$ . Este tipo de bijeção é comumente chamado de bijeção com preservação de arestas, de acordo com a noção geral de isomorfismo, sendo uma bijeção de preservação-de-estrutura.

A figura 2.2 representa um exemplo de isomorfismo entre dois grafos,  $G$  e  $H$ .

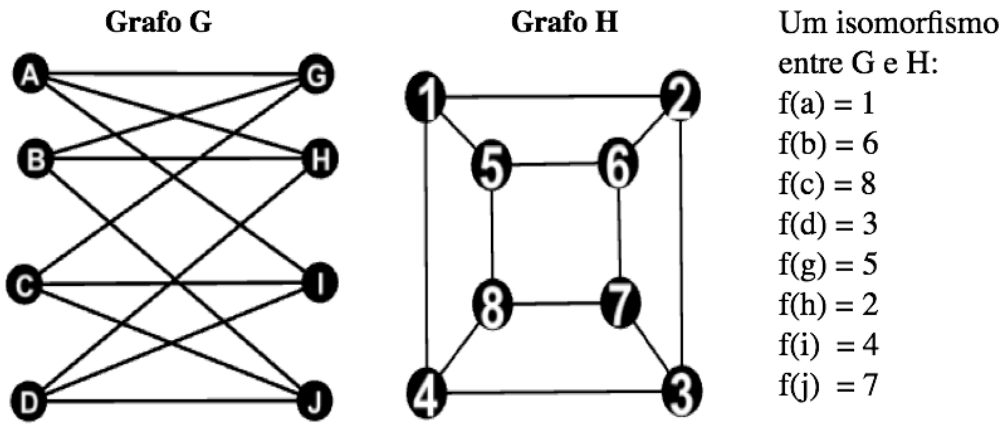


Figura 2.2: Isomorfismo em grafos.

## 2.2 Trabalhos relacionados

Nesta seção serão descritos dois trabalhos relacionados. Isto é, trabalhos que também estudam ataques em grafos anonimizados e detecção dos mesmos. Algumas das propostas presentes nestes trabalhos são utilizadas ao longo deste trabalho.

### 2.2.1 *Anonymized social networks, hidden patterns, and structural steganography*

Este trabalho é detalhado em [2], sendo o principal objetivo descrever uma família de ataques que, mesmo a partir de uma única cópia de uma rede social anonimizada, torna possível para o adversário aprender se existem relações entre usuários específicos. O trabalho desenvolvido neste artigo foi a inspiração para o nosso trabalho.

Três ataques diferentes são propostos e avaliados, sendo dois ativos e um passivo. Para isso, são considerados dados de redes sociais na forma mais pura, em que nós representam indivíduos e arestas indicam a interação social entre eles, de maneira

não direcionada. Estes dados estão anonimizados, isto é, o identificador de cada usuário é trocado por um identificador aleatório.

Assim como no nosso trabalho, o trabalho proposto sugere dois processos diferentes para promover o ataque: a criação dos usuários atacantes, no caso dos ataques ativos, e a recuperação do subgrafo criado. Este último processo também pode ser chamado de identificação dos nós atacantes, utilizando a informação estrutural dos nós atacantes para encontrá-los no grafo anonimizado.

### **Ataque passivo**

Neste ataque, usuários da rede tentam descobrir suas posições no grafo anonimizado usando o conhecimento que possuem sobre a estrutura local da rede em torno deles. Neste trabalho, os autores imaginaram uma pequena aliança formada por atacantes passivos, usuários da rede, que se uniram para se identificar na rede anonimizada que foi divulgada. Ao fazerem isso, eles comprometem a privacidade de seus vizinhos: aqueles conectados a um subconjunto único de usuários que formam a aliança serão reconhecidos unicamente após a identificação dos usuários que estão realizando o ataque. Neste caso, os nós que formam a aliança compõem o subgrafo a ser encontrado no grafo anonimizado.

Uma aliança  $X$  de tamanho  $k$  é iniciada por um usuário que recruta  $k - 1$  de seus vizinhos para se juntar à aliança. Os autores assumem que os usuários que formam a aliança conhecem as arestas entre eles, além de assumir que eles conhecem as identificações dos seus vizinhos que não estão presentes em  $X$ .

Resultados empíricos mostram que a aliança é facilmente encontrada no grafo anonimizado a partir da informação estrutural que ela possui, isto é, informações dos próprios nós que fazem parte da aliança, como grau. Entretanto, isto não significa que o algoritmo é capaz de identificar os demais usuários com certeza. Uma aliança de tamanho  $k$  não é capaz de comprometer mais do que  $2k - 1$  usuários e, na prática, os autores perceberam que o número é muito menor do que este. Isso se deve ao fato dos nós vizinhos à aliança nem sempre estarem conectados de maneira única aos nós que formam a aliança, de modo que mesmo após todos os nós atacantes terem sido reconhecidos ainda não é possível identificar unicamente seus vizinhos.

### **Ataques ativos**

Os ataques ativos descritos nesta subseção fazem uso de dois tipos de operação. Primeiro, o promotor do ataque pode criar novos usuários no sistema, adicionando novos nós ao grafo. Segundo, um nó  $u$  criado pelo adversário pode decidir se comunicar com um nó  $v$  do grafo, adicionado uma aresta não-direcionada  $(u, v)$ . O objetivo do ataque é determinar se é possível encontrar relações entre um conjunto

arbitrário de usuários. Esse conjunto possui tamanho representado pela letra  $b$  e é formado pelos nós vítimas, representados por  $w_i$ , sendo  $i$  o índice do nó.

Antes da rede anonimizada ser divulgada, o adversário cria  $k$  novos usuários e cria arestas entre eles, gerando um subgrafo  $H$ . O próximo passo é utilizar essas novas contas criadas para adicionar conexões com os nós pertencentes ao conjunto de nós vítimas,  $w_1, \dots, w_b$  e conectar o subgrafo criado com o grafo a ser atacado.

O promotor do ataque tenta encontrar a cópia de  $H$  no grafo anonimizado e, a partir disto, identificar os nós  $w_1, \dots, w_b$ . Se o promotor do ataque tiver identificado corretamente todos os nós vítimas, então ele é capaz de revelar as relações entre eles. Assim como no nosso trabalho, cada nó vítima está conectado a um subconjunto único de nós atacantes, porém com tamanho máximo definido pelo promotor do ataque.

É importante ressaltar que o subgrafo  $H$  é gerado pelo atacante sem que ele conheça o grafo sendo atacado. Isto dificulta o processo, uma vez que ele precisa criar um subgrafo que seja identificado de maneira única, independentemente da estrutura da rede sendo atacada. Isto é, o subgrafo  $H$  não pode possuir isomorfismos com nenhum outro subgrafo gerado a partir do grafo anonimizado e nenhum automorfismo.

Dois ataques distintos são descritos nesta subseção do artigo. Para o primeiro, os autores mostram que para  $k = \theta(\log n)$  novos usuários, um subgrafo  $H$  gerado aleatoriamente será único com alta probabilidade, independentemente das propriedades estruturais do grafo anonimizado e de como  $H$  é anexado a  $G$ , o grafo inicial que está sendo atacado. Além disso, se o grau máximo de  $H$  for  $\theta(\log n)$ , então  $H$  é recuperado eficientemente, junto da identificação dos nós vítimas, sendo  $b = \theta(\log^2 n)$ . Este ataque é capaz de comprometer a privacidade de  $\theta(k^2)$  usuários. O segundo ataque é similar ao primeiro, porém possui diferenças na maneira como o subgrafo  $H$  é anexado ao grafo  $G$  original, uma vez que usa menos arestas para conectá-los. A fina ligação entre  $H$  e  $G$  implica que o subgrafo criado provavelmente será único e eficientemente encontrado em um valor assintoticamente menor de  $k$ : este ataque utiliza  $k = \mathcal{O}(\log n)$  para revelar a identidade de  $\mathcal{O}(\sqrt{\log n})$ .

No trabalho de Lars é provado que os algoritmos de ataque ativo irão funcionar, com alta probabilidade, desde que não existam automorfismos não triviais no subgrafo  $H$  criado.

O ataque proposto por este trabalho, por sua vez, tem como objetivo criar  $\mathcal{O}(\log n)$  novos usuários e ser capaz de revelar a identidade de  $\mathcal{O}(n)$  nós.

### 2.2.2 *Resisting structural re-identification in anonymized social networks*

Este trabalho é apresentado em [5], em que os autores identificam riscos de privacidade associados a liberação de conjuntos de dados de uma rede e fornecem um algoritmo que mitigue esses riscos. Manter a privacidade ao publicar um conjunto de dados de rede é excepcionalmente desafiador porque o contexto da rede de um indivíduo pode ser usado para identificá-los, mesmo que outras informações de identificação sejam removidas, assim como é feito em [2]. No nosso trabalho utilizamos informações estruturais da rede, adicionadas pelo promotor do ataque, para identificar nós de uma rede anonimizada.

O foco deste artigo é analisar os resultados de divulgações resultantes da re-identificação estrutural ou topológica, em que a informação do adversário é apenas sobre a estrutura do grafo. Além disso, ele fornece uma avaliação abrangente dos riscos de privacidade da anonimização ingênua, que os autores descrevem como sendo apenas a omissão da identificação de cada usuário ao divulgar uma rede.

A partir da anonimização ingênua, o trabalho propõe um modelo adversário baseado nas assinaturas estruturais, que se baseia em uma classe de consultas de conhecimento, de poder crescente, que relata a estrutura local do grafo em torno de um nó. Essas consultas são inspiradas pelo refinamento iterativo de vértices [6] e são denotadas por  $H_i$ , em que  $H_0$  retorna a identificação do nó (em grafos anonimizados não retorna nada),  $H_1(x)$  retorna o grau do nó  $x$ ,  $H_2(x)$  retorna a sequência de grau dos vizinhos de  $x$  e assim por diante. As consultas podem ser definidas iterativamente, em que  $H_i(x)$  retorna os conjuntos de valores que são o resultado da avaliação de  $H_{i-1}$  no conjunto de nós adjacentes à  $x$ . O trabalho presente utiliza as consultas  $H_1$  e  $H_2$  para fazer o reconhecimento dos nós adicionados ao grafo atacado.

Os autores definem um conjunto de candidatos a partir de uma consulta  $H_i$ . Este conjunto é formado por todos os nós do grafo que possuem o mesmo valor  $H_i$ . Por exemplo, no caso de  $H_1$ , todos os nós com mesmo grau ficam no mesmo conjunto. A ideia é que nós em um mesmo conjunto são irreconhecíveis de acordo com  $H_i$ , precisando de mais informações para tentar identificar unicamente esses nós. É analisado, para 7 redes, reais e geradas a partir de modelos matemáticos, a quantidade de nós identificados a partir de  $H_1$ ,  $H_2$ ,  $H_3$  e  $H_4$ . Como quanto maior o índice da consulta feita mais informações o atacante possui sobre a rede, a probabilidade de identificar um nó corretamente cresce de  $H_1$  para  $H_2$ , de  $H_2$  para  $H_3$  e assim por diante.

Automorfismos fornecem segurança contra ataques como este, uma vez que a identificação única de alguns nós se torna impossível. O trabalho analisa que na

maior parte das redes é esperado encontrar classes de automorfismos pequenas, de modo a não ser suficiente para realizar a proteção contra ataques deste tipo.

O artigo também encontra que o fator crítico que determina o risco de re-identificação em grandes grafos aleatórios, gerados a partir de modelos matemáticos, é a densidade (a fração das arestas que existem no grafo sobre todas as possíveis arestas que poderiam existir). Grafos esparsos (com densidade baixa) têm baixo risco de re-identificação estrutural, enquanto os grafos densos têm alto risco.

Além disso, uma técnica de anonimização é proposta para proteger grafos contra ataques de re-identificação a partir de sua estrutura. O grafo é generalizado a partir do agrupamento de nós em partições. Ao em vez de divulgar o grafo anonimizado que representa exatamente a estrutura da rede em questão, é publicado o número de nós em cada partição juntamente com a densidade de arestas que existem dentro e entre as partições. Desta forma, o adversário tenta fazer a re-identificação, sem sucesso, enquanto o analista utiliza o grafo para estudar as propriedades do grafo original. Um algoritmo, chamado de GraphGen, é proposto, em que dois parâmetros são passados: o grafo original e um valor numérico representado pela letra  $k$ . O grafo anonimizado possuirá super nós, também chamados de partições, com pelo menos  $k$  nós da rede original. É também proposta uma extensão a este algoritmo, sendo a ideia apenas anonimizar nós vulneráveis a re-identificação, de modo a preservar mais a estrutura da rede. É feita uma avaliação extensa dos algoritmos propostos.

# Capítulo 3

## Algoritmo proposto

### 3.1 Explicação detalhada

A explicação em alto nível já foi feita no capítulo de introdução. Em cada subseção a seguir um componente específico será explicado detalhadamente.

#### 3.1.1 Definições matemáticas

Ao longo deste capítulo serão utilizadas diversas definições matemáticas para melhor explicar o cenário sendo discutido. Portanto, nesta subseção são declaradas todas as definições a serem utilizadas.

1. Grafo  $G(V, E)$  original, pré-ataque.
  - (a)  $V \equiv$  conjunto de vértices deste grafo.
  - (b)  $E \equiv$  conjunto de arestas deste grafo.
  - (c)  $n = |V|$ , o número de nós neste grafo.
2.  $X \equiv$  conjunto de vértices atacantes, isto é, a serem adicionados ao grafo.
  - (a)  $x = |X|$ , o número de nós atacantes.
  - (b)  $X = \{a_0, a_1, \dots, a_{x-1}\}$ .
3.  $H \equiv$  subgrafo induzido pelos nós atacantes. Isto é, subgrafo composto pelos nós atacantes e arestas entre eles.
  - (a)  $E_H \equiv$  arestas presentes no subgrafo  $H$ .
4.  $Y \equiv$  conjunto dos vértices vítimas, isto é, vértices sendo atacados.
  - (a)  $Y \subseteq V$ .
  - (b)  $y = |Y|$ , o número de nós vítimas.



5.  $Z \equiv$  conjunto de vértices não vítimas  $\subseteq V$  que não foram escolhidos pelo promotor do ataque e se conectam a vértices atacantes  $\in X$ .
  - (a)  $z = |Z|$ , o número de nós não vítimas.
6.  $G_a \equiv$  grafo final anonimizado e disponibilizado.
  - (a)  $V_a = V \cup X$ .
  - (b)  $E_a = E \cup E_H \cup E_{XY} \cup E_{XZ}$ , sendo,  $E_{XY}$  as arestas entre nós dos conjuntos  $X$  e  $Y$  e  $E_{XZ}$  as arestas entre nós dos conjuntos  $X$  e  $Z$ .
7.  $s_i(a_i) \equiv$  sequência de grau dos vizinhos do atacante  $a_i$ .
8.  $s'_i(a_i) \equiv$  subsequência de  $s_i$  formada somente pelos vizinhos de  $a_i$  que são atacantes. Esta é a sequência de grau dos vizinhos de  $a_i$  conhecida, uma vez que o promotor do ataque não possui informações a respeito do grau de nós  $\in V$ . Ele conhece o grau apenas dos nós  $\in X$ .
9.  $c_a(y_i) \equiv$  subconjunto de nós atacantes conectados ao nó vítima  $y_i$ .  $c_a(y_i) \subseteq X$ .
10.  $c_a(Y) = \bigcup_{i \in Y} c_a(y_i)$

Para exemplificar, os dados apresentados na figura 1.1 são utilizados para demonstrar os valores das definições apresentadas:

- $V = \{0, 1, 2, 3\}$ .
- $E = \{(1, 2), (2, 3), (0, 2), (0, 3), (1, 3)\}$ .
- $n = 4$ .
- $X = \{4, 5, 6, 7\}$ .
- $x = 4$ .
- $E_H = \{(4, 5), (5, 6), (6, 7), (4, 7), (5, 7)\}$ .
- $Y = \{0, 1\}$ .
- $y = 2$ .
- $Z = \{2, 3\}$ .
- $z = 2$ .
- $E_{XY} = \{(0, 4), (0, 5), (1, 6), (1, 7)\}$ .
- $E_{XZ} = \{(3, 6), (2, 5)\}$ .

- $s_i(4) = \{4, 5, 4\}$ .
- $s'_i(4) = \{5, 4\}$ .
- $c_a(0) = \{4, 5\}$ .
- $c_a(1) = \{6, 7\}$ .
- $c_a(Y) = \{\{4, 5\}, \{6, 7\}\}$

### 3.1.2 Criação dos nós atacantes

Este processo recebe como entrada  $G(V, E)$  e  $Y$ , sendo  $1 \leq y \leq n$ .

#### Número de atacantes criados ( $x$ )

Conforme explicado sucintamente na introdução,  $|c_a(y_i)| \geq 2, \forall y_i \in Y$ . Além disso,  $y_i \neq y_j \rightarrow c_a(y_i) \neq c_a(y_j), \forall y_i, y_j \in Y$ , uma vez que o processo de identificação dos nós vítimas em  $G_a$  será feito por meio do reconhecimento dos nós atacantes. Dessa forma, a maneira como cada nó vítima se relaciona aos nós atacantes é única e se for possível reconhecer unicamente os nós pertencentes a  $X$ , será possível reconhecer unicamente todos os nós vítimas, uma vez que o atacante sabe exatamente as relações existentes entre esses dois conjuntos de nós ( $E_{XY}$ ).

O tamanho mínimo de  $c_a(y_i), \forall y_i \in Y$  ser 2 é uma premissa definida pelo método. De modo a tornar o subgrafo de atacantes criado (formado por  $H \cup E_{XY} \cup E_{XZ}$ ) mais aleatório e evitar a presença de isomorfismos entre este subgrafo e qualquer subgrafo gerado a partir de  $G_a$ , o processo tenta criar arestas entre  $Z$  e  $X$ . Para isso, é preciso que nós  $\in Z$  se conectem a subconjuntos de  $X$  que  $\notin c_a(Y)$ , para que não sejam confundidos por um nó vítima. Dessa forma, o modelo proposto define que os nós não atacados só podem se conectar a um nó atacante. O objetivo desta prática é evitar que o algoritmo precise, a cada vez que gerar um subconjunto de  $X$ , procurar se ele  $\notin c_a(Y)$ .

Sendo assim,  $\min(|c_a(Y)|) = y$ . Para gerar todos os possíveis subconjuntos de  $X$ , basta elevar 2 a  $X$ : cada nó pertencente ao conjunto pode ou não estar no subconjunto, sendo duas possibilidades, resultando em  $2^X$ . Como  $|c_a(y_i)| \geq 2, \forall y_i \in Y$ , é necessário remover todos os subconjuntos de tamanho igual a 1 (existem  $x$  subconjuntos neste caso) e o subconjunto com tamanho igual a 0, isto é, aquele que não possui nenhum atacante, só existindo 1 subconjunto que representa este caso.

$$y = 2^x - 1 - x$$

Sabendo que  $y$  pode variar entre 1 e  $n$ , será utilizado o pior caso para calcular o valor de  $x$ , que é quando o promotor do ataque deseja violar a privacidade de todos os nós do grafo, isto é,  $y = n$ .

Logo:

$$n \leq 2^x - 1 - x$$

Sabendo que  $n$  e  $x$  são números inteiros positivos:

$$n \leq 2^x - 1 - x < 2^x$$

$$n < 2^x$$

$$x > \lceil \log_2 n \rceil$$

Além disso, o propósito deste processo é não precisar criar muitos atacantes para conseguir que o ataque seja sucedido. Conhecendo o limitante inferior, o objetivo será encontrar um limitante superior que não faça com que o número de atacantes seja um número alto quando comparado a  $n$ . O objetivo é encontrar um limitante superior que possua complexidade  $\mathcal{O}(\log n)$ . Desta forma, sendo  $c$  uma constante:

$$x \leq c * \log_2 n$$

$$x \leq c * \log_2(2^x - 1 - x) < c * \log_2(2^x)$$

$$x < c * \log_2(2^x)$$

$$x < c * x$$

$$0 < (c * x) - x$$

Consequentemente:  $c > 1$

Neste trabalho será utilizado  $c = 2$ , por ser o menor inteiro possível a ser utilizado.

$$X = 2 * \lceil \log_2 n \rceil$$

Logo, os  $2 * \lceil \log_2 n \rceil$  novos nós da rede são controlados pelo ataque, que conhece as informações estruturais deles, uma vez que ele quem criou esses nós e as arestas incidentes a eles.

### Criação das arestas

Para o ataque funcionar, é preciso encontrar o subgrafo  $H$  no grafo  $G_a$ . Portanto, este subgrafo (e as arestas incidentes aos seus nós) será criado com propriedades

específicas, de modo a tentar se sobressair do restante da rede. Como o algoritmo não conhece  $G(V, E)$ , o subgrafo gerado possuirá sempre as mesmas propriedades.

Uma vez que os nós atacantes já foram adicionados ao grafo, falta adicionar as arestas envolvendo estes nós.

1. Criação das arestas entre os atacantes

- (a) Toda aresta do tipo  $(a_i, a_{i+1})$  será criada. Isto é, o  $a_0$  estará ligado ao  $a_1$ , o  $a_1$  estará ligado ao  $a_2$  e assim por diante, fechando um ciclo entre todos os atacantes, uma vez que a aresta  $(a_0, a_{x-1})$  também é criada. Este ciclo é utilizado na busca dos nós atacantes no grafo anonimizado, que será discutida mais a frente.

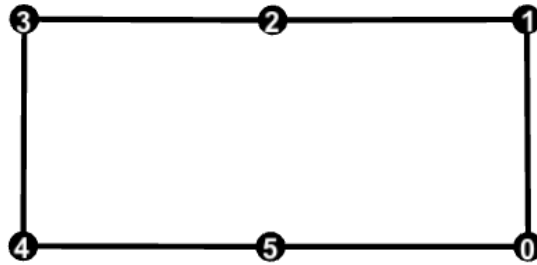


Figura 3.1: Exemplo do subgrafo de atacantes após adição inicial de arestas.

- (b) As demais arestas serão criadas aleatoriamente: a respectiva aresta entre cada par de nós  $(a_i, a_j)$ , sendo  $i < (j + 1)$ , existe com probabilidade  $1/2$ . A escolha deste modelo matemático para gerar as arestas aleatórias, com probabilidade  $1/2$  de existir uma aresta entre dois nós, foi feita uma vez que deste modo todos os possíveis grafos a serem gerados têm a mesma probabilidade de acontecer.

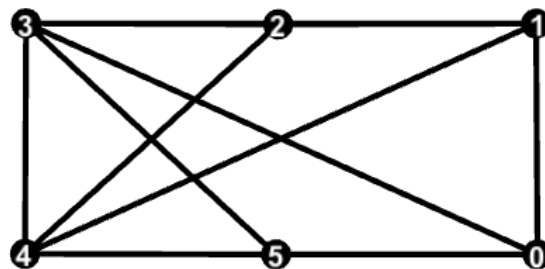


Figura 3.2: Exemplo do subgrafo de atacantes após adição das arestas aleatórias entre os atacantes.

2. Criação das arestas entre nós atacantes e nós vítimas ( $E_{XY}$ ): ao final deste passo, todos os nós vítimas precisam estar conectados à um subconjunto único de nós atacantes de tamanho maior igual a dois.

Para gerar os subconjuntos  $c_a(y_i)$  com tamanho maior igual a dois para cada  $y_i \in Y$ , foi criado um vetor de booleanos de tamanho igual ao número de atacantes, que será chamado de vetor de marcação. Cada posição do vetor só pode assumir dois valores: 0 (falso) ou 1 (verdadeiro). Esse vetor é inicializado com todas as posições contendo o valor 0 e será incrementado iterativamente.

Para cada nó vítima, o processo verifica se o vetor de marcação possui ao menos duas posições preenchidas com o valor 1. Se possuir: para cada posição que estiver com o valor 1, uma aresta é criada entre o atacante correspondente (índice no vetor de onde está o valor 1) e o nó vítima em questão. Em seguida, o vetor é transformado em um número binário de 0's e 1's e é adicionado 1 unidade binária à esse número, que é então transformado em um vetor de marcação novamente. Caso não possua pelo menos duas posições com o valor 1, apenas a soma binária acontece. Esse processo é repetido para cada nó vítima. A transformação entre vetor de booleanos e número binário é feita da maneira como se segue neste exemplo: vetor  $[1,0,1,1,1]$  se transforma no número 10111 e vice-versa.

Dessa forma é garantido que todo nó vítima estará conectado a pelo menos dois atacantes e que nenhum outro nó vítima estará conectado à exatamente os mesmos nós atacantes, tendo em vista que o vetor é alterado a cada iteração, sendo sempre diferente.

A tabela 3.1 representa o vetor de marcação em um exemplo ao longo de algumas iterações. Cada coluna representa uma posição do vetor: como existem 4 atacantes, o vetor possui tamanho igual a 4 e cada posição do vetor está associada a um atacante. Por exemplo, a posição 0 do vetor está associada ao  $a_0$ , e assim por diante.

Para fins demonstrativos, apenas as iterações iniciais são mostradas na tabela. Para cada nó vítima, o objetivo é encontrar uma iteração do vetor de marcação com mais de uma posição preenchida com o valor verdadeiro. Logo, o processo é iniciado com o primeiro nó vítima do conjunto, que será representado por  $y_0$ . Nas iterações 1, 2 e 3 apenas a soma binária de uma unidade acontece, pois não existem posições suficientes com o valor 1. Na iteração 4, entretanto, existem, de modo que o  $y_0$  é conectado aos nós  $a_2$  e  $a_3$ . Em seguida, o próximo nó vítima, representado por  $y_1$ , é trabalhado e a soma binária acontece. Na iteração 5 apenas a soma acontece e na 6 o nó  $y_1$  é conectado aos nós  $a_1$  e  $a_3$ . O processo continua até todos os nós vítimas terem se conectado aos atacantes dessa forma.

Tabela 3.1: Vetor de marcação de atacantes a cada iteração, com 4 atacantes.

Iteração	0	1	2	3
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1

A figura 3.3 representa um fluxograma explicando resumidamente o que foi discutido no passo 2 das criações de arestas, que é a formação das arestas entre nós atacantes e nós vítimas. Cada nó vítima é relacionado a um subconjunto único de nós atacantes, de tamanho maior que 1.

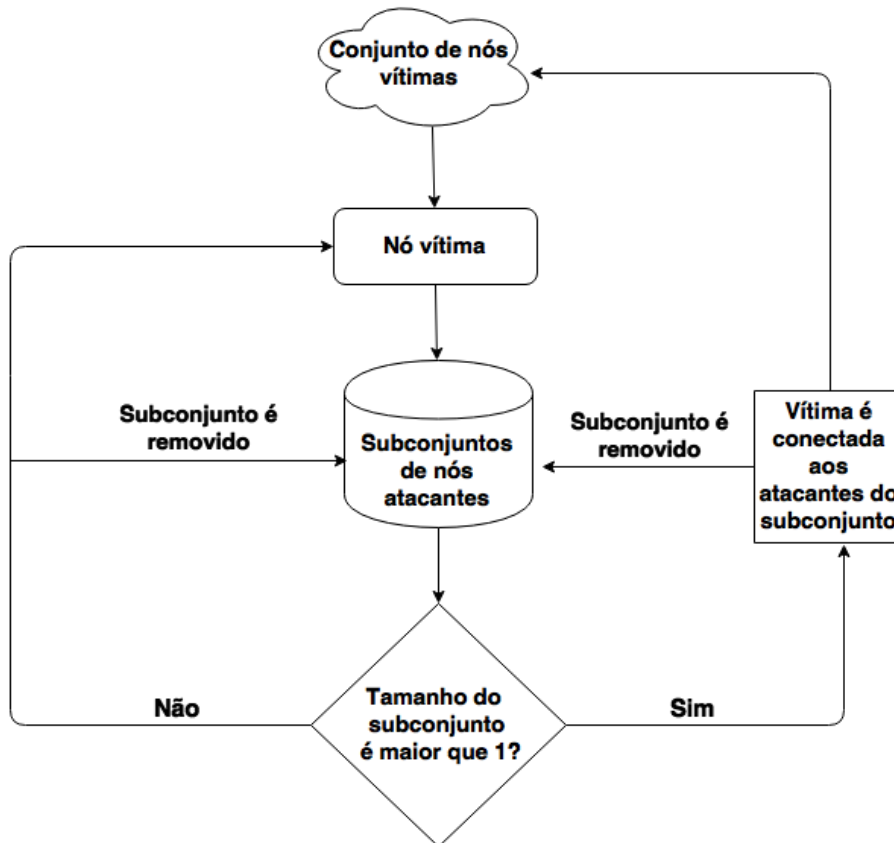


Figura 3.3: Fluxograma explicando as ligações entre nós atacantes e nós vítimas.

3. Por fim, o algoritmo cria arestas entre nós atacantes e nós não vítimas ( $E_{XZ}$ ).

Esse processo é feito de modo a aumentar a aleatoriedade do subgrafo criado pelo promotor do ataque e diminuir as chances de encontrar um isomorfismo dele no grafo  $G_a$ . Essa parte será discutida detalhadamente mais a frente.

Neste passo, é importante diferenciar os conceito de graus internos e externos dos nós atacantes: grau interno é a quantidade de nós atacantes ao qual o nó está relacionado ( $|s'_i(a_i)|$ ) e grau externo é a quantidade de nós não atacantes, isto é,  $\in V$ , ao qual um nó atacante está relacionado. A soma dos graus internos e externo é o grau do nó atacante ( $|s_i(a_i)|$ ).

Para cada nó atacante, um grau externo máximo é gerado a partir de uma distribuição uniforme entre o grau externo que o nó possui, isto é, a quantidade de nós atacados aos quais ele está conectado, e um número representado pela letra  $d$ . O valor de  $d$  varia de acordo com o grau externo do nó: se o grau externo for igual a zero, isto é, aquele atacante não está conectado a nenhum nó  $\in V$ , então  $d = x$ . Caso contrário,  $d$  segue a seguinte fórmula:

$$d = [((rand() \pmod{x}) + 1) * ((rand() \pmod{x}) + 1)] + g_e$$

, em que  $g_e$  representa o valor do grau externo do nó em questão e  $rand()$  é a função de c++ que retorna um número aleatório entre 0 e 32767, que foi inicializada com a semente sendo igual a  $x$ . A figura 3.4 é a representação gráfica da geração de grau externo máximo de cada nó atacante.

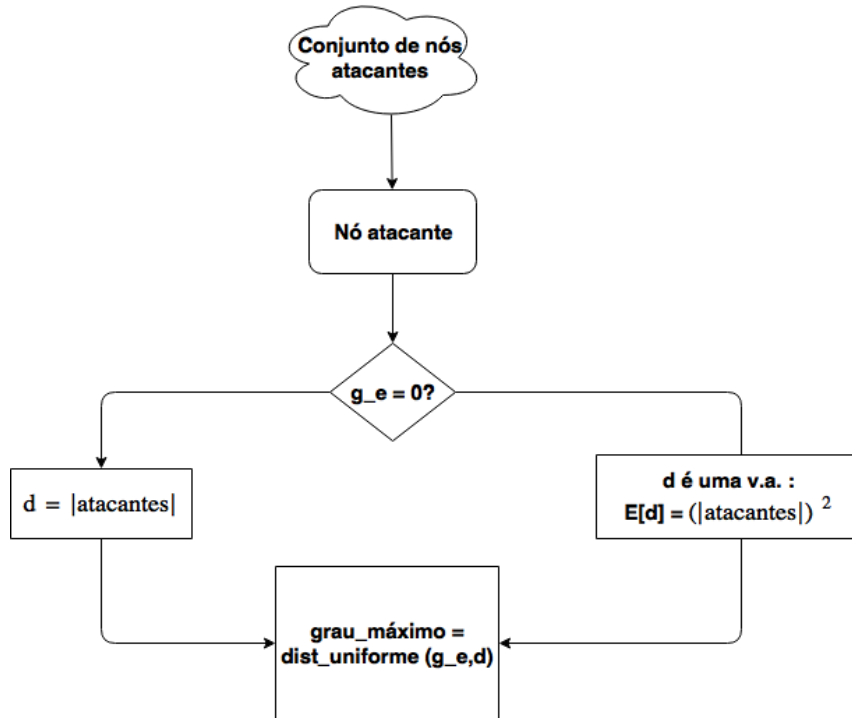


Figura 3.4: Fluxograma explicando a geração de grau externo máximo.

Após cada atacante receber um grau máximo externo, o próximo passo é tentar criar relações entre nós atacantes e não vítimas para atingir este valor de grau externo máximo definido. É necessário, porém, apenas conectar nós não vítimas a subconjuntos de atacantes  $\notin c_a(Y)$ , de modo a não confundir, no processo de reconhecimento no grafo anonimizado, com um nó vítima. Para facilitar esse processo e acelerá-lo, uma vez que não será necessário verificar quais subconjuntos já foram usados, o método só associa cada nó não vítima a um único nó atacante.

É necessário, aqui, gerar uma lista com todos os nós não vítimas. Cada nó desta lista só poderá ser conectado a um nó atacante, respeitando a regra definida no parágrafo acima. Assim sendo, para cada nó atacante, enquanto ele não tiver atingido seu grau máximo externo e o tamanho dessa lista for maior que zero, o algoritmo escolhe aleatoriamente um nó não vítima da lista, o removendo da lista e o adicionando a  $Z$ , e o associa ao atacante em questão, aumentando em uma unidade seu grau externo. Esse processo é repetido para todos os nós atacantes até conseguir alcançar todos os graus máximos externos gerados ou não existirem mais nós não vítimas na lista. No caso do promotor do ataque decidir atacar todos os nós da rede, por exemplo, não existem nós não vítimas e o grau externo final de cada nó é igual ao grau externo calculado após o fim do passo 2 descrito. A figura 3.5 representa graficamente a ideia de como os nós atacantes e os nós não vítimas são relacionados.



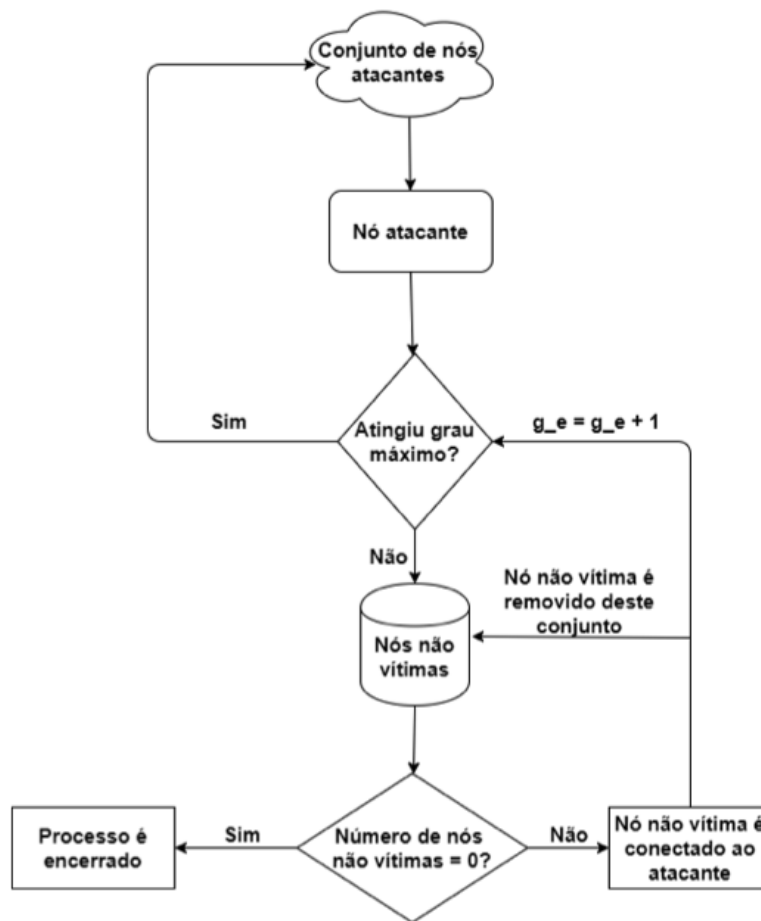


Figura 3.5: Fluxograma explicando as ligações entre nós atacantes e nós não vítimas.

### Verificação de Automorfismos

Por fim, o algoritmo faz uma busca simples para verificar a possível existência de automorfismos no subgrafo de atacantes criado. Não é possível afirmar, entretanto, se existem automorfismos, pois seria necessário conhecer toda a estrutura da rede formada após os passos acima, como grau de todos os nós da rede. O atacante só possui informações sobre os nós que ele mesmo adicionou, como o grau e os seus vizinhos, então essas informações serão usadas para verificar a existência de possíveis automorfismos. Se o algoritmo identificar um possível automorfismo, então todo o processo descrito até então neste capítulo é refeito. Existe um número de tentativas máximas por execução para refazer este processo, de modo a evitar possíveis, porém improváveis, infinitas iterações.

Para o processo final, de identificação dos nós vítimas ou reconhecimento do subgrafo de atacantes, funcionar corretamente, o subgrafo de atacantes criado não pode possuir automorfismos ou isomorfismos com qualquer outro subgrafo gerado a partir de  $G_a$ . Caso exista, não será possível identificar de maneira única todos os nós atacantes no grafo anonimizado, de modo que alguns nós atacantes se tornem

irreconhecíveis. Como os nós vítimas são identificados a partir de suas relações com os nós atacantes, também não será possível identificá-los.

Não é possível, durante a criação dos atacantes, identificar isomorfismos entre o subgrafo criado e qualquer subgrafo formado a partir de  $G_a$ , uma vez que a estrutura de  $G$  é desconhecida. Um isomorfismo entre qualquer subgrafo da rede e o subgrafo de atacantes significaria confundir pelo menos um atacante com um não atacante. Como o ataque não possui qualquer informação estrutural sobre os não atacantes, nós  $\in V$ , e o reconhecimento de isomorfismos depende dessa informação, não é possível procurar por isomorfismos neste componente. Deste modo, o algoritmo busca evitar isomorfismos tentando gerar subgrafos de atacantes predominantemente aleatórios, tornando improvável a existência de configurações locais da rede iguais às configurações locais do subgrafo gerado.

Pelo mesmo motivo, também não é possível afirmar se existem ou não automorfismos. Porém, é viável procurar por possíveis automorfismos, de modo a tentar evitá-los.

Supondo que toda a estrutura da rede fosse conhecida, o problema para encontrar automorfismos em um grafo é um problema NP-difícil. O processo criado neste trabalho para procurar por possíveis automorfismos, pelo contrário, possui uma complexidade polinomial e foi inspirado na ideia proposta em [5]: utilizar o grau de cada nó e a sequência de grau dos vizinhos de cada nó ( $s_i(a)$ ).

Desse modo, a ideia consiste em utilizar todas as informações que o promotor do ataque possui sobre a estrutura da rede, isto é, a informação que ele mesmo adicionou: o grau de cada nó atacante. Utilizando essa informação, também é possível pensar na sequência de grau dos vizinhos de cada atacante ( $s_i(a_i)$ ). Essa sequência de grau dos vizinhos, entretanto, será parcial: só terá informação sobre os vizinhos que também são atacantes, uma vez que o grau dos nós  $\in V$  não é conhecido. Conforme dito no início deste capítulo, esta subsequência é representada por  $s'_i(a_i)$ .

Uma condição necessária para existir um automorfismo é dois nós atacantes,  $a_i$  e  $a_j$ , possuírem o mesmo grau, caso contrário eles jamais seriam confundidos a partir de sua estrutura. Outra condição necessária, que só é analisada caso a primeira condição seja verdadeira, é  $s_i(a_i) = s_i(a_j)$ . Como o processo não tem acesso à  $s_i$ , a condição neste trabalho é os graus de  $a_i$  e  $a_j$  serem iguais e  $s'_i(a_i) \subset s'_i(a_j)$  ou  $s'_i(a_j) \subset s'_i(a_i)$ . Ambas condições descritas neste parágrafo são necessárias, porém não são suficientes. Além dessas condições, existem outras também necessárias para garantir um automorfismo. Como o trabalho está interessado em um algoritmo rápido para identificar apenas a possibilidade de existir um automorfismo, apenas as condições descritas são usadas.

Para exemplificar a segunda condição descrita, suponha dois nós atacantes, com

valor de grau igual a 5. O primeiro nó,  $a_0$ , está conectado a três atacantes e duas vítimas, de modo que  $s'_i(a_0)$  tenha tamanho 3 e seja igual a, por exemplo,  $[1,2,3]$ . O segundo nó,  $a_1$ , por sua vez, está conectado a quatro atacantes e uma vítima, de modo que  $s'_i(a_1) = [1, 2, 3, 4]$ , por exemplo. Para finalizar, é preciso supor o grau dos três nós vítimas conectados a  $a_0$  e  $a_1$ . Neste exemplo, suponha que o grau deles é igual a 4. Dessa forma,  $s_i(a_0) = s_i(a_1) = [1, 2, 3, 4, 4]$ , podendo ocorrer uma confusão entre esses dois nós no grafo anonimizado e, conseqüentemente, um automorfismo. Este exemplo mostra um possível automorfismo a partir das condições descritas e das informações que ele possui.

O algoritmo, por sua vez, cria um dicionário, em que a chave é um número, o grau do nó, e o valor é uma lista de  $s'_i(a_i), \forall a_i \in X$  com grau igual à chave. Para cada atacante  $a_i$ , gera-se  $s_i(a_i)$ , e verifica se para o grau deste atacante já existe uma sequência igual a dele, que contenha a dele ou que ele contém. Se não existir, adiciona a sequência no dicionário na lista que possui o grau dele como chave e vai para o próximo atacante. Caso exista, o processo acusa um possível automorfismo, e, se ele não tiver tentado gerar um subgrafo de atacantes mais vezes do que o limite estipulado, todo o processo descrito nesta seção é repetido. A figura 3.6 apresenta um fluxograma que resume como a verificação de automorfismos, descrita neste parágrafo, funciona.

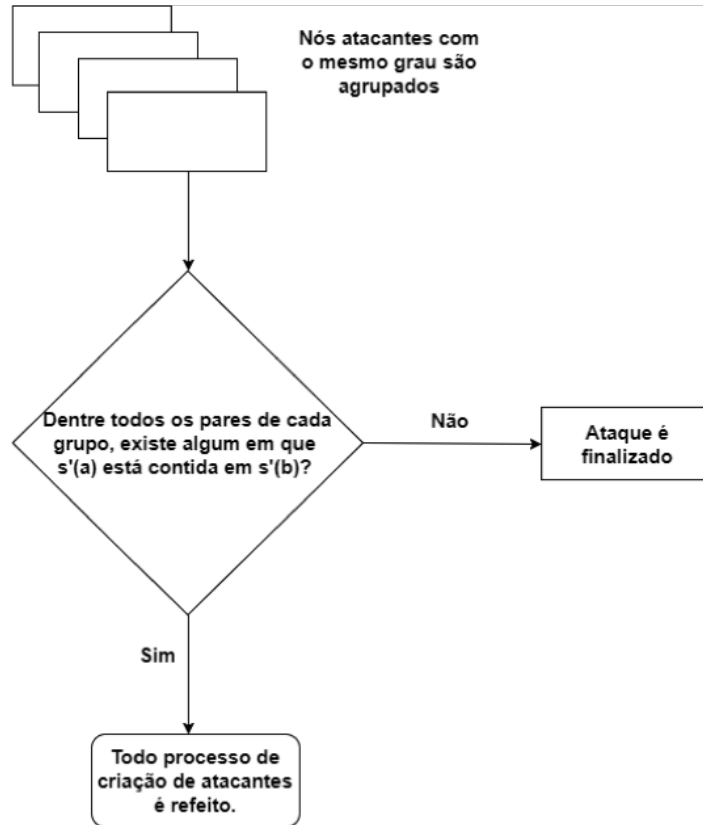


Figura 3.6: Fluxograma explicando a verificação de automorfismos.

Vale ressaltar que a acusação de um possível automorfismo não significa que obrigatoriamente vá existir um automorfismo, uma vez que apenas uma parte da sequência de grau dos vizinhos não é suficiente para afirmar a existência de um automorfismo. Dessa forma, esse processo analisa uma condição necessária, porém não suficiente.

A complexidade deste processo varia com o número de atacantes, igual a  $x$ , e o número de vizinhos de cada atacante. No pior caso, um atacante está conectado ao grafo inteiro e a todos os nós atacantes, isto é,  $n + x - 1$ . Como  $x = 2 * \lceil \log_2 \rceil n$ , a complexidade é igual a  $\mathcal{O}((\log_2 n) * (n + \log_2 n))$ , que é igual a  $\mathcal{O}(n * \log_2 n)$ .

### Arquivos de Saída

Por fim, existem dois arquivos que são gerados a partir deste processo e são utilizados nos próximos: o grafo  $G'(V', E')$ , em que  $V' = V \cup X$  e  $E' = E \cup E_H \cup E_{XZ} \cup E_{XY}$ , criado por este componente, e um arquivo com as informações estruturais dos atacantes, isto é, o grau de cada atacante e  $s'_i(a_i)$ .

### 3.1.3 Anonimização do grafo

A proposta deste trabalho é identificar nós escolhidos pelo promotor do ataque, também chamados de nós vítimas, de modo a revelar relações entre eles. Para ser possível analisar a eficiência do algoritmo proposto, isto é, se o mesmo é capaz de identificar os nós vítimas ou não, é necessário conhecer a identificação de cada nó na rede, de modo que seja possível comparar o resultado encontrado para cada nó com a informação original.

Para isso, este componente fará a anonimização do grafo, de modo a simular o processo que a empresa dona da rede a ser divulgada faria antes de disponibilizar os dados.

Esse processo recebe como entrada a rede gerada pelo processo anterior, isto é, o grafo  $G'(V', E')$ , em que  $V' = V \cup X$  e  $E' = E \cup E_H \cup E_{XZ} \cup E_{XY}$ . A saída dele, por sua vez, são dois arquivos: a rede anonimizada  $G_a$ , que possui a mesma estrutura da rede  $G'(V', E')$  porém com os identificadores anonimizados que foram gerados, sendo um isomorfismo de  $G'$ ; o mapeamento de cada nó, ou seja, um mapa entre a identificação que cada nó possuía na rede de entrada e o novo identificador anônimo gerado. Este arquivo será utilizado para analisar a eficiência do método no final, ao tentar identificar os nós vítimas, comparando a identificação encontrada para cada nó com a identificação real.

- Para a geração dos identificadores, todos os identificadores reais de cada nó são dispostos em uma lista e existem duas opções para gerar a troca de identidade.

Para exemplificar este processo, a figura 3.7 representa um grafo com 5 nós e arestas não-direcionadas entre eles que será anonimizado.

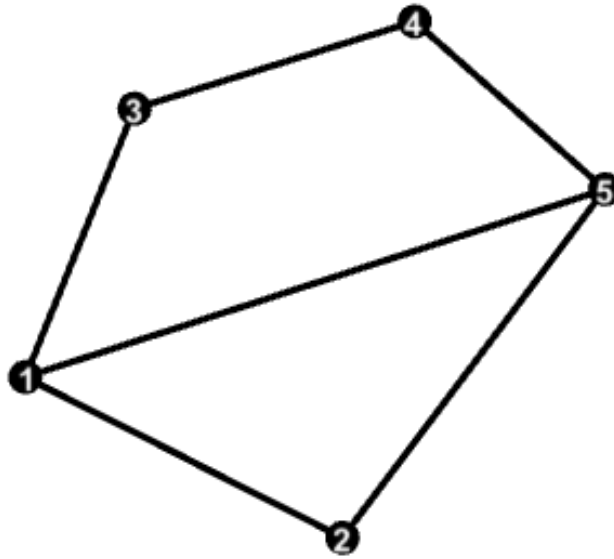


Figura 3.7: Grafo para exemplificar componente de anonimização.

1. Todos os identificadores originais são adicionados em uma lista, de maneira aleatória, e cada nó recebe um novo identificador aleatório, que é o índice do seu identificador real nesta lista gerada

Tabela 3.2: Resultados do processo de anonimização a partir do índice de cada nó.

Identificador Real	Identificador Anonimizado
2	4
3	2
4	0
1	3
5	1

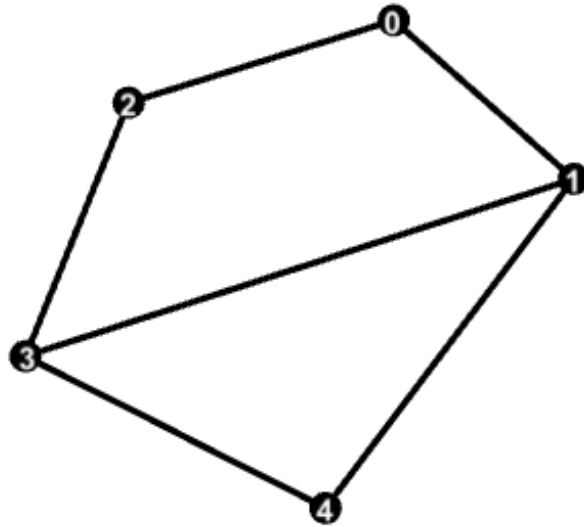


Figura 3.8: Grafo anonimizado pelo processo de anonimização a partir do índice de cada nó.

2. Cada nó recebe o identificador de outro nó, isto é, ocorre uma permutação entre os identificadores existentes.

Tabela 3.3: Resultados do processo de anonimização a partir da permutação entre os identificadores.

Identificador Real	Identificador Anonimizado
4	5
3	3
2	4
1	2
5	1

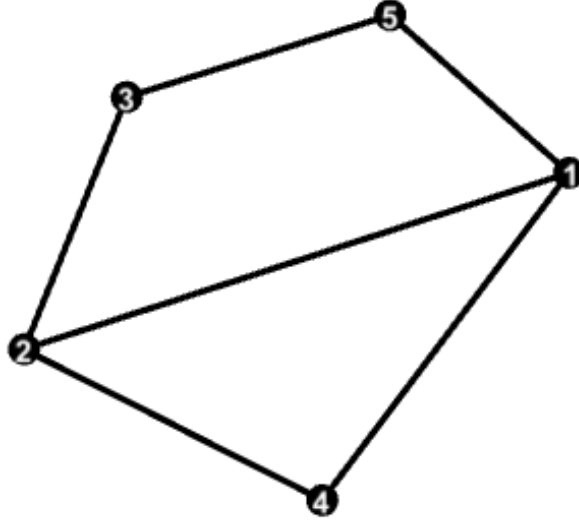


Figura 3.9: Grafo anonimizado pelo processo de anonimização a partir da permutação entre os identificadores.

Como este processo é aleatório, diferentes execuções com os mesmos parâmetros podem gerar arquivos de saída diferentes. A fim de exemplificação, as tabelas e imagens representadas são possíveis resultados extraídos do algoritmo de acordo com a geração dos identificadores.

### 3.1.4 Recuperação do subgrafo

O objetivo deste componente é encontrar o subgrafo de atacantes  $H$  criado no componente Criação de Atacantes. Pela definição do algoritmo, se for possível identificar todos os atacantes criados, então todos os nós vítimas, que foram os nós definidos pelo promotor do ataque para terem sua privacidade violada, serão identificados. Consequentemente, relações entre estes nós também serão reveladas, que é o objetivo do algoritmo.

Este componente recebe de entrada os seguintes arquivos: o grafo anonimizado,  $G_a$ , gerado no processo de anonimização do grafo descrito; o arquivo contendo a informação de cada atacante, isto é, informação de grau e  $s'_i(a_i)$ ; o subgrafo de atacantes criado,  $H$ , com todas as arestas incidentes à esses nós ( $E_{XY}$  e  $E_{XZ}$ ), de modo a identificar os nós vítimas após a identificação dos atacantes; o arquivo que mapeia o identificador real de cada nó para o identificador anonimizado, para em caso de falha ser possível descobrir se o motivo foi isomorfismo ou automorfismo.

O primeiro passo é carregar todas as informações passadas por arquivos de texto para a memória do computador, em estruturas de dados eficientes para o propósito de cada informação.

Em seguida, este componente irá procurar, no grafo anonimizado, pelos nós

atacantes, que são aqueles que o ataque possui alguma informação estrutural sobre. Para isso, o processo irá procurar em ordem pelos atacantes, isto é, irá procurar pelo  $a_0$ , depois pelo  $a_1$  e assim por diante. De modo a encontrar os nós atacantes, o processo irá utilizar uma lista de árvores. Cada caminho na árvore representará um possível subgrafo  $H$ .

Para encontrar  $a_0$ , o processo passa por todos os nós  $\in G_a$  e analisa quais nós possuem o mesmo grau que o atacante sendo buscado. Para todo nó  $u \in G_a$  que tiver o mesmo grau de  $a_0$ , o processo então analisa se  $s'_i(a_0) \subseteq s_i(u)$ , uma vez que o promotor do ataque não conhece  $s_i(a_0)$ , por não conhecer o grau dos vértices  $\in V$ . Se  $s'_i(a_0) \subseteq s_i(u)$ , então uma árvore é criada e adicionada à lista de árvores, em que o vértice  $u$  será a raiz. Isto indica que o vértice  $u \in G_a$  é um possível  $a_0$  anonimizado. Este processo é repetido até verificar todos os nós  $\in G_a$ . A figura 3.10 representa a busca pelo primeiro atacante apresenta graficamente, em que cada nó candidato ao atacante em questão vira um nó raiz de uma árvore.

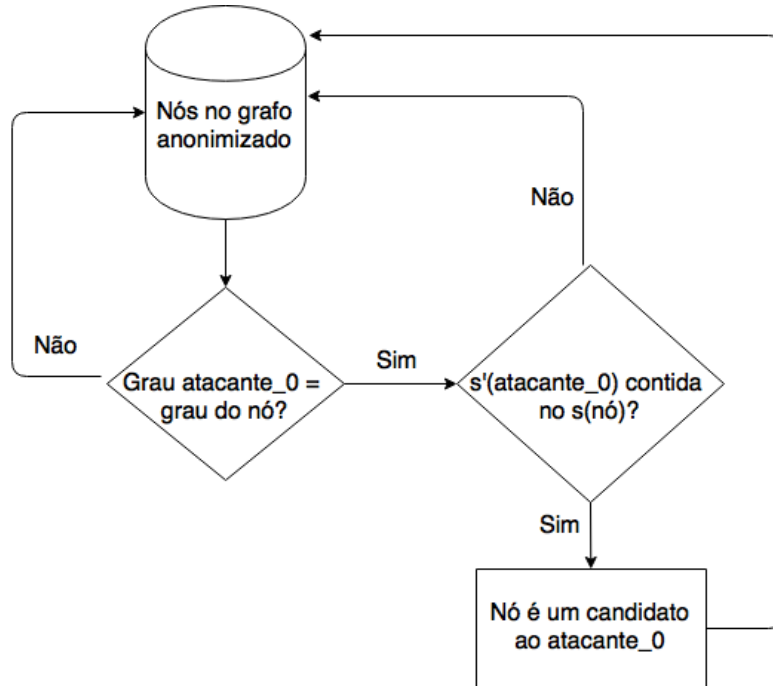


Figura 3.10: Fluxograma explicando a busca pelo nó atacante inicial.

Para os demais nós atacantes, o algoritmo irá passar por cada árvore e, nestas, irá trabalhar com cada nó folha. A altura da árvore determina o atacante que o nó anonimizado pode representar. Por exemplo, o nó que está na altura 2, é um possível  $a_2$ .

Dessa forma, para cada atacante (diferente do  $a_0$ ), o algoritmo irá agir sobre cada nó folha de cada árvore. Cada nó folha representa um possível  $a_i$ , sendo  $i$  a altura na árvore em que esse nó se encontra. O processo irá procurar, utilizando o mesmo procedimento descrito para identificar o  $a_0$ , o próximo nó atacante. Isto



é, se o nó folha está representando um possível  $a_i$ , então o algoritmo irá procurar dentre todos os vizinhos do nó folha, no grafo  $G_a$ , pelo  $a_{i+1}$ . Vale lembrar que de acordo com a maneira que os atacantes foram criados, os  $a_i$  e  $a_{i+1}$  estão conectados, para  $0 \leq i \leq (x - 1)$ . A figura 3.11 representa por meio de um fluxograma a ideia apresenta.

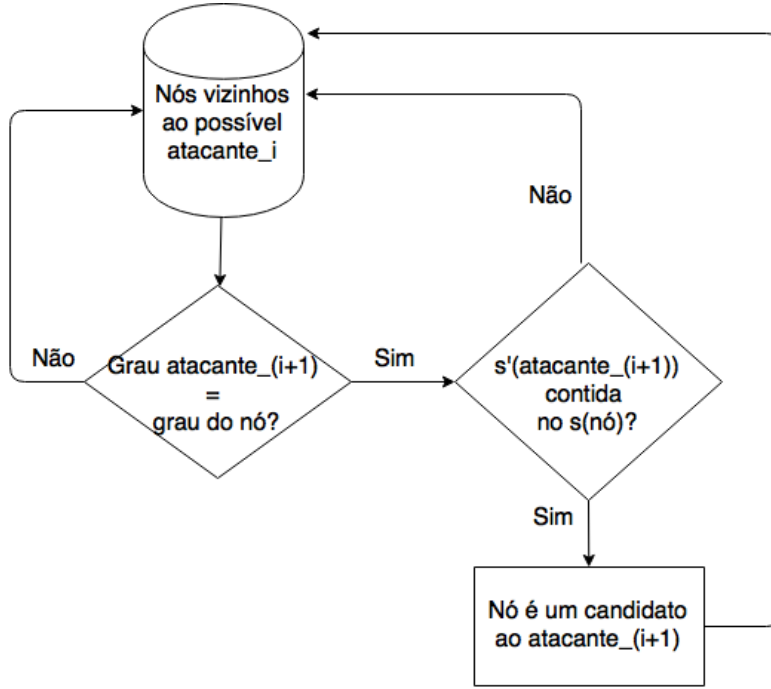


Figura 3.11: Fluxograma explicando a busca pelos demais nós atacantes.

A figura 3.12 é um exemplo de possíveis árvores geradas após 3 iterações de busca por atacantes. Existem dois possíveis nós para serem o  $a_0$ ,  $B$  e  $G$ , que possuem o mesmo grau deste atacante e  $s'_i(a_0) \subseteq s_i(B)$  e  $s'_i(a_0) \subseteq s_i(G)$ . Em seguida, o processo irá procurar nos vizinhos de  $B$ , no grafo  $G_a$  pelo  $a_1$  e encontra duas possibilidades: os nós anonimizados  $D$  e  $A$ . O mesmo processo é feito na árvore com a raiz  $G$ , que encontra dois possíveis  $a_1$ :  $E$  e  $H$ . Na terceira iteração, quando o  $a_2$  está sendo procurado, dentre os vizinhos de  $D$ ,  $E$  e  $H$  nenhum tinha as condições necessárias para representar o  $a_2$ , de modo que o processo só tenha encontrado um possível  $a_2$  dentre os vizinhos de  $A$ . Como os  $a_1$  e  $a_2$  estão obrigatoriamente conectados, os nós  $D$ ,  $E$  e  $H$  não podem ser o  $a_1$ . Vale ressaltar que  $(A, B, C, D, E, F, G, H)$  são identificadores anonimizados de cada nó.

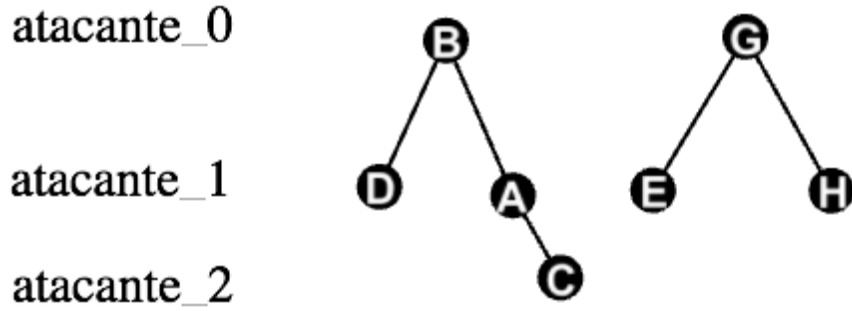


Figura 3.12: Árvore exemplificando a recuperação dos atacantes no grafo anonimizado.

Se durante uma iteração um nó folha representando um possível  $a_i$  não encontrar em seus vizinhos um possível  $a_{i+1}$ , o nó folha em questão é apagado, uma vez que ele não pode representar o  $a_i$  já que por definição ele está conectado ao  $a_{i+1}$ . Uma vez apagado este nó folha, o algoritmo analisa se o pai deste nó na árvore virou folha. Se sim, ele também é apagado e o processo continua subindo na árvore para o pai deste, até encontrar um nó não folha ou a raiz. Caso contrário, nada é feito a mais neste passo e o processo continua normalmente. Deste modo, ao final de cada iteração, só existem nós folhas na altura da iteração - 1, que representa o número do possível nó atacante.

O processo de busca pelos atacantes continua até uma das seguintes situações acontecerem: todos os possíveis nós para atacantes foram encontrados, isto é, existe pelo menos uma árvore com altura igual a  $x - 1$  (considerando que a altura da árvore começa em zero); a soma de nós folhas em todas as árvores ultrapassou um limite estipulado pelo promotor do ataque. Isto acontece para que o processo não gere milhares de possíveis subgrafos de atacantes, o que ocorre quando existem muitos automorfismos no subgrafo de atacantes ou isomorfismos no grafo anonimizado.

Uma vez que o processo tenha terminado, por qualquer dos motivos citados acima, o próximo passo é analisar se o algoritmo foi sucedido ou não.

### Sucesso do Algoritmo

O algoritmo terá sido bem sucedido se ao final da execução do processo descrito ele encontrar uma única árvore, de altura igual a  $x - 1$  (considerando que a altura inicial da árvore é igual a 0) e com  $x$  nós. Isto é, um grafo linha com  $x$  nós, em que cada nó representa um nó atacante em ordem: a raiz representa o  $a_0$ , o nó conectado à raiz representa o  $a_1$  e assim por diante. Se isso acontecer, então o processo funcionou e todos os nós vítimas e as relações entre eles podem ser identificadas. Caso contrário, o processo falhou. A figura 3.13 representa um caso em que o ataque foi bem sucedido e todos os nós vítimas podem ser identificados. Os identificadores  $A$ ,  $B$ ,  $C$ ,  $H$  e  $X$

são os identificadores anônimos de cada nó atacante, em ordem.

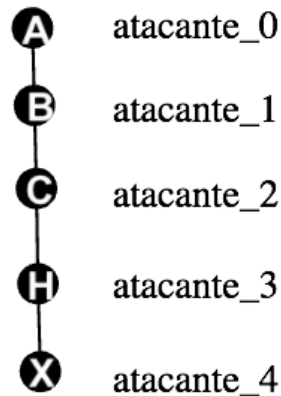


Figura 3.13: Árvore representando o sucesso na recuperação dos atacantes no grafo anonimizado.

### Falha do Algoritmo

O algoritmo falha se ele não for capaz de identificar todos os atacantes no grafo anonimizado, isto é, se a árvore gerada, no final do processo, não representar um grafo linha com  $x$  nós.

- Só existem dois possíveis motivos para o processo falhar:
  1. Apesar da aleatoriedade do processo, ainda ocorreu pelo menos um isomorfismo entre o subgrafo formado pelos nós atacantes e todas as arestas incidentes à eles e um subgrafo qualquer gerado a partir de  $G_a$ . Dessa forma, um ou mais nós não atacantes são confundidos com nós atacantes. A figura 3.14 representa esse caso, em que não é possível diferenciar o nó atacante 2 de um nó pertencente ao grafo original.

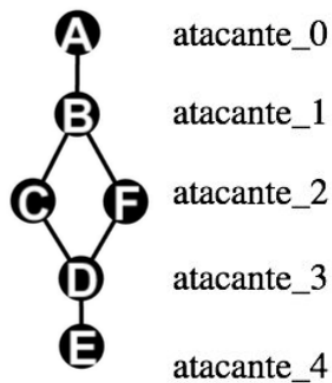


Figura 3.14: Exemplo de árvore representando falha causada por isomorfismo.

2. Automorfismo não identificado do subgrafo, isto é, não é possível diferenciar um par ou mais de atacantes. Por exemplo, dado um nó qualquer de  $G_a$ , pelas informações que o processo possui, ele pode ser tanto o  $a_0$  quanto o  $a_1$ . Apesar do processo de criação de nós e arestas procurar por automorfismos e evitá-los, ele não conhece a estrutura do restante da rede, de modo que ele não é capaz de evitar por completo a existência de automorfismos. A figura 3.15 representa esse caso.

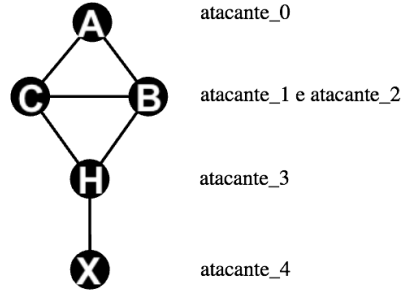


Figura 3.15: Exemplo de árvore representando falha causada por automorfismo.

Para exemplificar o caso em que o processo de verificação de automorfismos não é capaz de acusar um possível automorfismo, suponha dois nós atacantes,  $a_i$  e  $a_j$ , com o mesmo grau ( $|s_i(a_i)| = |s_i(a_j)|$ ) e o mesmo grau interno ( $|s'_i(a_i)| = |s'_i(a_j)|$ ). Porém,  $s'_i(a_i) \not\subseteq s'_i(a_j)$  e  $s'_i(a_j) \not\subseteq s'_i(a_i)$ :  $s'_i(a_i) = [1, 2, 3]$  e  $s'_i(a_j) = [1, 2, 4]$ , ou seja, o algoritmo de verificação de automorfismo não é capaz de identificar um automorfismo aqui. Supondo que o grau desses nós atacantes seja igual a 4, cada um deles só estaria conectado a um nó vítima. Nesse sentido,  $a_i$  está conectado a  $y_0$ , que possui grau igual a 4 e  $y_1$  está conectado a  $y_0$  que possui grau igual a 3. Portanto,  $s_i(a_i) = s_i(a_j) = [1, 2, 3, 4]$ , o que caracteriza um possível automorfismo, porém que não foi identificado pelo processo. Apenas isso não é suficiente para afirmar que de fato ocorrerá um automorfismo, dado que as informações dos nós vizinhos a cada um deles também importam. Existe, entretanto, uma outra possibilidade para a presença de um automorfismo: o processo de verificação de automorfismos funciona, de modo a refazer todo o processo de criação de nós atacantes e suas arestas. Porém, existe um número máximo de vezes, escolhido pelo promotor do ataque, que o processo é capaz de refazer tudo. Se o processo gerar mais possíveis automorfismos identificáveis do que o número máximo de vezes permitido, então o processo de recuperação do subgrafo poderá falhar, uma vez que o processo de criação dos atacantes foi concluído com um subgrafo de atacantes com um possível automorfismo detectado.

Por fim, na avaliação do algoritmo é utilizado o mapeamento de cada identificador real para o identificador anônimo para verificar o motivo da falha, isto é, verificar se ocorreu um isomorfismo ou automorfismo.

## 3.2 Implementação

Os algoritmos descritos neste capítulo foram implementados na linguagem de programação c++ e nenhuma biblioteca externa foi utilizada.

Como nenhum algoritmo complexo de grafos foi utilizado, não foi utilizada uma biblioteca externa para a representação de grafos. Apenas estruturas nativas da linguagem, como *unordered map*, *unordered set*, listas e dicionários foram utilizados para o armazenamento eficiente das informações sobre os nós e suas relações. Cada processo utiliza as estruturas que melhor funcionam para si, isto é, que diminuem a complexidade de execução do algoritmo.

A cada execução do algoritmo os grafos são passados como arquivos de texto, em que cada linha possui dois identificadores, separados por espaço, que representa uma aresta relacionando esses dois nós representados. Os grafos são então salvos em memória e os algoritmos são executados.

Para a automatização dos testes, *scripts* foram desenvolvidos na linguagem *shell script*, de modo a possibilitar a execução de diferentes grafos, repetidas vezes. Esses *scripts* executam os três processos descritos neste capítulo de maneira sequencial, em que a saída de um dos componentes é a entrada do próximo.

Por fim, *scripts* em python também foram desenvolvidos para a realização de algumas tarefas, como o gerador de grafos do modelo de grafos aleatórios de Erdős-Rényi [7], em que dois parâmetros são passados: o número de nós do grafo e a probabilidade de existir uma aresta entre quaisquer dois nós. Este modelo de grafos é usado para analisar a eficiência do método no capítulo 4.

Todo código desenvolvido neste trabalho pode ser encontrado no *github* em <https://github.com/pamtabak/GraphPrivacy>.

# Capítulo 4

## Avaliação

### 4.1 Metodologia da avaliação

Como discutido no capítulo 3, de forma a possibilitar a avaliação dos resultados gerados pelo algoritmo desenvolvido, foram utilizadas redes que não são anonimizadas nesta seção, de modo que este trabalho simule a anonimização da rede após a criação dos atacantes. Caso o processo obtenha sucesso, pela definição do algoritmo, não seria necessário conhecer a rede inicial, uma vez que se foi encontrado apenas um subgrafo de atacantes, é possível identificar os nós vítimas, sem a informação de mapeamento entre os identificadores reais e os anonimizados. Entretanto, se o algoritmo falhar, esse mapeamento serve para identificar o motivo da falha, isto é, se foi um problema de isomorfismo ou automorfismo. Neste caso, existe pelo menos algum nó atacante que não foi unicamente reconhecido, isto é, possui mais de um possível nó anonimizado. Para identificar o tipo de falha, basta buscar pelo identificar desses possíveis nós representando o atacante e checar se ele é um outro nó atacante (automorfismo) ou um nó do grafo original (isomorfismo).

Todas as redes apresentadas na seção de resultados, portanto, são redes reais ou redes aleatórias criadas a partir de modelos matemáticos de redes. Para cada rede e para cada número de nós vítimas, que pode variar entre um e o número de nós do grafo, foram feitas 50 execuções. Na execução do processo de criação de atacantes, o número máximo de vezes que o processo pode encontrar um possível automorfismo e, portanto, refazer todo o processo de geração do subgrafo formado pelos nós atacantes é igual a 10. O número máximo de nós folhas que as árvores podem ter, no processo de recuperação do subgrafo criado, é igual a 10000.

Além disso, apenas grafos não-direcionados foram utilizados para a avaliação do algoritmo. Como em grafos direcionados existem mais informações sobre cada nó (por exemplo: graus de entrada e de saída ao em vez de apenas um grau), o processo de encontrar nós a partir da estrutura da rede se torna mais fácil nos grafos

direcionados. Ademais, o algoritmo foi desenvolvido especialmente para grafos não-direcionados, de modo a não fazer distinção entre o sentido de uma aresta. Vale ressaltar que o algoritmo pode ser remodelado de modo a aceitar redes direcionadas e, então, analisar a eficácia do método.

## 4.2 Resultados

Pelo fato do processo de recuperação do subgrafo precisar carregar o grafo inteiro em memória e para a avaliação do ataque possuir mais dados, isto é, muitas execuções para diferentes redes e número de nós sendo atacados, o número de nós das redes utilizadas nesta seção se encontra na casa dos milhares.

Os resultados dispostos nesta seção representam a quantidade de sucessos por rede e número de nós vítimas, normalizado pelo número de execuções, de modo a variar entre 0 e 1. No caso de ocorrerem falhas, resultados demonstrando o tipo de falha, isomorfismo ou automorfismo, também são apresentados.

### 4.2.1 Grafos aleatórios

Esta subseção irá apresentar o desempenho do algoritmo quando executado em duas redes geradas a partir de modelos matemáticos, em que suas estruturas topológicas são aleatórias, isto é, a formação de arestas segue regras de formação com aleatoriedade.

#### Erdős-Rényi [7]

O modelo Erdős-Rényi, mais especificamente o modelo  $G(n, p)$ , é um modelo matemático que recebe dois parâmetros: o número de nós que a rede deve ter, representado pela letra  $n$ , e a probabilidade de existir uma aresta entre dois nós, representada pela letra  $p$ , que pode variar entre 0 e 1. Cada aresta é incluída no grafo de maneira independente e com mesma probabilidade, igual a  $p$ .

Para a execução deste processo, foram utilizados 1000 nós para a rede e diferentes valores de  $p$ , variando entre 0.01 a 0.5. Conforme o valor de  $p$  aumenta, o número de arestas aumenta consideravelmente: no caso em que  $p$  vale 0.5, com 1000 nós na rede, existem 250 mil arestas. Por esse motivo, não foram utilizados valores de  $p$  maiores que 0.5.

Para cada combinação de  $p$  e número de nós vítimas, 50 execuções foram feitas, gerando uma nova rede seguindo o modelo aleatório descrito em cada uma delas. Foram escolhidas 152 combinações diferentes de valores para a probabilidade  $p$  de existir uma aresta entre dois nós e número de nós vítimas, de modo a serem feitas 7600 execuções deste processo no total, representadas na tabela 4.1. É possível,

portanto, reparar que apenas uma execução não foi sucedida de todas as executadas: um erro ao atacar todos os nós da rede com probabilidade de existir uma aresta entre dois nós igual a 1%.

Este erro foi causado por um isomorfismo, isto é, pelo menos um nó não atacante foi confundido com um nó atacante. Para entender o que aconteceu, é preciso analisar a rede em que o ataque não foi sucedido: um modelo de rede de Erdős-Rényi com 1000 nós e probabilidade de existir uma aresta entre dois nós igual a 1%. No caso, todos os nós da rede estão sendo atacados e existem 20 nós atacantes ( $2 \log_2 1000$ ). Além disso, o valor esperado do grau,  $g$ , dos nós neste modelo de rede é igual a  $E[g] = n * p$ , sendo igual a 10 nesta rede específica. Pela maneira como os nós atacantes são conectados aos nós vítimas, descrita no capítulo 3, apenas parte dos atacantes são relacionados a vítimas: neste caso, apenas metade dos atacantes possui arestas com nós vítimas. Assim, a outra metade está apenas conectada com os próprios atacantes, uma vez que não existem nós não vítimas para eles se conectarem, já que a rede inteira está sendo atacada. Como cada nó atacante se conecta aos demais atacantes com probabilidade igual a 50% (ignorando o fato dele se conectar aos dois atacantes mais próximos de maneira determinística), é possível calcular o valor esperado do grau dos atacantes que não possuem arestas com os nós vítimas da mesma maneira que foi calculado para o grafo inicial: multiplicar o número de atacantes pela probabilidade de existir uma aresta entre eles, ou seja,  $20 * 0.5$ , encontrando um valor igual a 10, igual ao valor esperado para o grau dos nós vítimas. É possível, portanto, perceber que existem muitos nós vítimas com valor esperado de grau muito próximo (e até mesmo igual) ao valor esperado de grau de alguns nós atacantes. Uma vez que o grau de um atacante é igual ao de uma vítima, é utilizada parte da sequência de grau dos vizinhos para tentar identificá-los, porém, dada a grande quantidade de nós com grau semelhante, é capaz que não seja suficiente e o isomorfismo aconteça.

Tabela 4.1: Desempenho do algoritmo no modelo Erdős-Rényi.

Nº Vítimas	p=0.01	p=0.05	p=0.08	p=0.1	p=0.2	p=0.3	p=0.4	0.5
2	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1
32	1	1	1	1	1	1	1	1
64	1	1	1	1	1	1	1	1
100	1	1	1	1	1	1	1	1



Tabela 4.1 – continuação

Nº Vítimas	p=0.01	p=0.05	p=0.08	p=0.1	p=0.2	p=0.3	p=0.4	0.5
128	1	1	1	1	1	1	1	1
200	1	1	1	1	1	1	1	1
256	1	1	1	1	1	1	1	1
300	1	1	1	1	1	1	1	1
400	1	1	1	1	1	1	1	1
500	1	1	1	1	1	1	1	1
512	1	1	1	1	1	1	1	1
600	1	1	1	1	1	1	1	1
700	1	1	1	1	1	1	1	1
800	1	1	1	1	1	1	1	1
900	1	1	1	1	1	1	1	1
1000	0.98	1	1	1	1	1	1	1

### Barabási-Albert (BA) [8]

Este modelo de rede aleatório segue uma regra muito antiga de formação de grupos, chamada de *preferential attachment* em inglês. A ideia principal é que objetos têm preferência em se relacionar com objetos mais populares. No caso de redes, os objetos são os vértices, as relações as arestas e a popularidade é dada pelo grau do vértice.

Este modelo de rede possui dois parâmetros principais: o número de nós do grafo, representado pela letra  $n$ , e o grau mínimo de cada nó, representado pela letra  $m$ . O processo de formação desse modelo de redes é feito de maneira iterativa, conforme descrito a seguir:

1. Inicialmente, um pequeno clique é formado, isto é, um grafo em que todos os vértices estão conectados entre si. O número de nós do clique é consideravelmente menor do que  $n$  e maior igual a  $m$ .
2. A cada iteração, um novo vértice com grau  $m$  é adicionado.
3. Os  $m$  vértices que terão arestas incidentes a este novo vértice, de modo que o seu grau seja igual a  $m$  ao final desta iteração, serão escolhidos aleatoriamente dentre os vértices já adicionados à rede, com probabilidade proporcional ao seus graus. Isto é, vértices com maior grau possuem maior chance de se relacionarem a este novo vértice.

Representando o número da iteração pela letra  $t$ , é possível descrever a probabilidade de um vértice  $u$ , já adicionado à rede, ser incidente à uma nova

aresta no instante  $t$  pela seguinte equação, em que  $d_u(t)$  é o grau do vértice  $u$  no instante  $t$  e a letra  $V$  representa o conjunto de vértices do grafo no instante  $t$ :

$$p_u(t) = \frac{d_u(t)}{\sum_{v \in V} d_v(t)} = \frac{d_u(t)}{2mt}$$

4. Repetir os itens 2 e 3 até que o grafo possua  $n$  nós.

O modelo Barabási–Albert (BA) gera redes sem escala, com distribuição de grau do tipo lei de potência, consequentemente tendo uma cauda pesada, propriedade amplamente observada em diversas redes complexas naturais e artificiais, como a web e algumas redes sociais.

Para a geração dos grafos utilizados para a avaliação do algoritmo proposto, este trabalho utilizou a biblioteca *graph-tool* na linguagem de programação python, que possui uma função que recebe os parâmetros  $m$  e  $n$  definidos pelo usuário e cria grafos não-direcionados respeitando o modelo descrito.

Foram escolhidos sete valores diferentes para o valor  $m$ , que representa o grau mínimo de cada nó, e dezenove valores para a quantidade de nós a serem atacados no grafo. Todos os grafos deste modelo criado possuem 1000 nós. Para cada possível configuração de teste, isto é, para cada combinação entre  $m$  e número de nós vítimas, foram feitas 50 execuções do algoritmo. A quantidade de sucessos está representada na tabela 4.2, que foi normalizada pela quantidade de execuções. Dessa forma, 6650 execuções diferentes foram feitas, das quais apenas 18 não foram sucedidas, tendo todas as falhas sido causadas por isomorfismo.

A partir da tabela 4.2, é possível perceber um certo padrão em relação aos erros: aproximadamente 83% deles ocorreram quando  $m$  é igual a 10 e os demais quando  $m$  é igual a 5, não ocorrendo erros nos demais valores de  $m$ . Como este modelo possui distribuição de grau com cauda pesada, existem muitos nós com grau baixo, próximo ao grau mínimo, e poucos nós com grau elevado - porém uma quantidade não desprezível de nós. Utilizando a mesma lógica da falha causada no modelo  $G(n, p)$ , quando  $m$  é igual a 10, existem muitos nós não atacantes com grau próximo dos graus de alguns atacantes. Dessa forma, ocorreram casos em que pelo menos um nó atacante  $a_i$  e um nó da rede original  $v_j$  tivessem o mesmo grau e a  $s'_i(a_i) \subseteq s_i(v_j)$ .

Tabela 4.2: Desempenho do algoritmo no modelo Barabási-Albert.

Nº Vítimas	m=5	m=10	m=20	m=30	m=40	m=50	m=100
2	1	1	1	1	1	1	1
4	1	0.98	1	1	1	1	1
8	1	1	1	1	1	1	1

Tabela 4.2 – continuação

Nº Vítimas	m=5	m=10	m=20	m=30	m=40	m=50	m=100
16	1	0.98	1	1	1	1	1
32	1	0.94	1	1	1	1	1
64	1	0.94	1	1	1	1	1
100	1	0.94	1	1	1	1	1
128	0.98	0.96	1	1	1	1	1
200	1	1	1	1	1	1	1
256	1	1	1	1	1	1	1
300	1	1	1	1	1	1	1
400	1	1	1	1	1	1	1
500	1	1	1	1	1	1	1
512	1	0.98	1	1	1	1	1
600	1	1	1	1	1	1	1
700	1	1	1	1	1	1	1
800	0.98	1	1	1	1	1	1
900	1	0.98	1	1	1	1	1
1000	0.98	1	1	1	1	1	1

### 4.2.2 Redes reais

Esta subseção irá apresentar o desempenho do algoritmo em três redes reais, retiradas da base de dados do projeto de análise de redes de Stanford, liderado por Jure Leskovec [9].

#### Facebook

Esta rede consiste de alguns círculos, ou listas de amigos, existentes no Facebook, coletada a partir de uma pesquisa de usuários que estavam usando o aplicativo desta rede social. A rede possui mais dados além de sua estrutura, porém este projeto só utiliza as informações de relacionamento entre os usuários.

Os dados foram anonimizados antes de serem publicados pelo grupo de Stanford em seu *website*, porém este trabalho considerou que os dados publicados são reais, de modo a realizar a comparação entre os resultados encontrados por este modelo e os dados da rede disponibilizada na plataforma.

A rede é conexa, possui 4039 nós e 88234 arestas, o diâmetro (maior caminho mínimo) vale 8 e o coeficiente de clusterização médio 0.6055. É interessante perceber que esta rede, mesmo sendo pequena, possui características comuns de redes sociais,

como alto valor de clusterização, baixo valor de diâmetro e é esparsa (densidade igual a 0.5%).

No total, 2600 execuções diferentes foram feitas. Cada linha da tabela 4.3 representa a quantidade de sucessos, normalizada pelo número de execuções, para um determinado número de nós vítimas. De todas as execuções feitas, apenas oito erros ocorreram, todos do tipo isomorfismo. O único caso em que dois erros ocorreram na mesma configuração, isto é, com o mesmo número de nós atacados, foi no caso que a rede inteira foi atacada. Nos demais casos, apenas um erro, ou nenhum, aconteceu.

Tabela 4.3: Desempenho do algoritmo em uma rede do Facebook.

<b>Nº Vítimas</b>	<b>Desempenho</b>
2	1
4	1
8	1
16	1
32	0.98
64	1
100	0.98
128	1
200	1
256	1
300	1
400	0.98
500	1
512	1
600	1
700	1
800	1
900	1
1000	1
1024	1
1100	1
1200	1
1300	1
1400	1
1500	1
1600	1

**Tabela 4.3 – continuação**

<b>Nº Vítimas</b>	<b>Desempenho</b>
1700	1
1800	1
1900	1
2000	1
2048	1
2100	1
2200	1
2300	1
2400	1
2500	1
2600	1
2700	1
2800	1
2900	1
3000	0.98
3100	1
3200	0.98
3300	1
3400	1
3500	0.98
3600	1
3700	1
3800	1
3900	1
4000	1
4039	0.96

### **Votação na Wikipédia**

A Wikipédia é uma enciclopédia gratuita escrita colaborativamente por voluntários ao redor do mundo. Uma pequena parte dos contribuidores desta rede social são administradores, que são usuários com acesso a recursos técnicos adicionais que auxiliam na manutenção. Para um usuário se tornar administrador é preciso que ele emita um pedido para a administração do site e, a comunidade da Wikipédia, por meio de uma discussão pública ou votação, decide quais usuários serão promovidos a administradores.

O grafo representa 2794 eleições, com um total de 103663 votos e 7066 usuários participando das eleições, isto é, todas as eleições que ocorreram desde o início da enciclopédia virtual até Janeiro de 2008. Nesta rede, os nós representam os usuários do sistema e as arestas são direcionadas e representam o voto, isto é, se existe uma aresta saindo do nó  $i$  e atingindo o nó  $j$ , então o usuário  $i$  votou no usuário  $j$ . Como este trabalho tem como principal objetivo estudar o desempenho do algoritmo em grafos não-direcionados, o grafo é transformado em não-direcionado, isto é, o sentido das arestas foi removido, de modo a apenas analisar a existência de uma aresta entre dois nós. Desta forma, o grafo trabalhado possui 7115 nós e 100762 arestas.

Ao todo, 1750 execuções diferentes foram feitas. Cada linha da tabela 4.4 representa a quantidade de sucessos, normalizada pelo número de execuções, para um determinado número de nós vítimas. Para os 35 valores de nós vítimas escolhidos, nenhum erro foi observado.

Tabela 4.4: Desempenho do algoritmo na rede de votação da Wikipédia.

<b>Nº Vítimas</b>	<b>Desempenho</b>
2	1
4	1
8	1
16	1
32	1
64	1
100	1
128	1
200	1
256	1
300	1
400	1
500	1
512	1
600	1
700	1
800	1
900	1
1000	1
1024	1
1500	1

**Tabela 4.4 – continuação**

<b>Nº Vítimas</b>	<b>Desempenho</b>
2000	1
2048	1
2500	1
3000	1
3500	1
4000	1
4096	1
4500	1
5000	1
5500	1
6000	1
6500	1
7000	1
7115	1

## **Gnutella P2P**

Esta rede é uma sequência de *snapshots* da rede de compartilhamento de arquivo P2P (*peer-to-peer*) de Agosto de 2002. No total, existem 9 *snapshots* da rede coletada nesse período de tempo. Como esta rede é extremamente dinâmica, o conceito de *snapshot* surgiu para capturar os dados de maneira estática, de modo que ele representa o estado da rede em um determinado tempo. Existem 6301 nós na rede, que representam os *hosts* da topologia de rede Gnutella e 20777 arestas, que representam as conexões entre os *hosts*. A rede é conexa, isto é, existe apenas uma grande componente conexa com todos os nós dentro.

Mais uma vez, os dados representam um grafo direcionado. De modo a utilizar o método criado e analisar a eficiência do mesmo, a direção das arestas foi removida, assim, se existia uma aresta saindo do nó  $x$  e chegando ao nó  $y$ , então existe uma aresta conectando os nós  $x$  e  $y$  sem direção no grafo a ser atacado.

A tabela 4.5 representa a quantidade de sucessos obtidos pelo algoritmo, normalizado pela quantidade de vezes que cada configuração foi executada. No total, foram realizadas 2200 execuções. É possível observar que apenas uma dessas execuções não foi sucedida, quando o número de vítimas foi igual a 6250. O erro causado foi, novamente, isomorfismo.

Tabela 4.5: Desempenho do algoritmo na rede de compartilhamento de arquivo Gnutella P2P.

<b>Nº Vítimas</b>	<b>Desempenho</b>
2	1
4	1
8	1
16	1
32	1
64	1
100	1
128	1
200	1
256	1
300	1
400	1
500	1
512	1
600	1
700	1
800	1
900	1
1000	1
1024	1
1250	1
1500	1
1750	1
2000	1
2048	1
2250	1
2500	1
2750	1
3000	1
3250	1
3500	1
3750	1
4000	1
4096	1
4250	1



**Tabela 4.5 – continuação**

<b>Nº Vítimas</b>	<b>Desempenho</b>
4500	1
4750	1
5000	1
5250	1
5500	1
5750	1
6000	1
6250	0.98
6301	1

# Capítulo 5

## Detecção do ataque

Como abordado na introdução, este trabalho se baseia na promoção de um ataque ativo, criando um subgrafo de novos usuários, chamados de atacantes. Isto é feito para que o promotor do ataque possua alguma informação sobre a estrutura da rede e possa encontrar, na rede anonimizada, os nós por ele criado e, conseqüentemente, identificar outros nós e suas relações.

Esta seção irá abordar a detecção do ataque, isto é, se seria possível identificar que usuários foram propositalmente inseridos na rede, com a intenção de atacá-la. Uma vez que o objetivo deste trabalho é, principalmente, focar no desenvolvimento e desempenho do ataque em si e não tanto em sua detecção, apenas uma técnica será utilizada. Esta, por sua vez, é a detecção de comunidades, muito conhecida no mundo de redes complexas e detalhadamente explicada no capítulo 9 do livro apresentado em [10].

### 5.1 Detecção de comunidade

O processo de detecção de comunidade tem como objetivo separar os vértices do grafo em grupos, conhecidos por *clusters*, a partir de características em comum entre os vértices. Cada grupo dá origem a uma comunidade. A ideia é que as comunidades são formadas por nós altamente conectados enquanto nós pertencentes a diferentes comunidades são fracamente conectados.

Para avaliar a detecção do ataque proposto por esse trabalho, o algoritmo de detecção de comunidades utilizado foi o disponibilizado no *software Gephi*, uma plataforma de visualização interativa, que foi utilizada para gerar todos os grafos apresentados neste trabalho, e exploração para todos os tipos de redes e sistemas complexos e dinâmicos.

O programa implementa o método de Louvain [11] para a detecção de comunidades, que utiliza a otimização de modularidade para encontrar comunidades.

### 5.1.1 Método de Louvain

#### Modularidade

Modularidade é uma função que mede a qualidade de uma partição, ou comunidade, representada pela letra  $Q$ , variando entre -1 e 1.

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$$

em que  $A_{ij}$  é o peso da aresta entre os nós  $i$  e  $j$ ,  $k_i$  é  $\sum_j A_{ij}$ ,  $m$  é igual a  $\frac{1}{2} \sum_{i,j} A_{ij}$ .  $c_i$  representa a comunidade à qual o vértice  $i$  pertence e  $\delta(c_i, c_j)$  vale 1 se  $c_i = c_j$  e 0 caso contrário. As variáveis  $i$  e  $j$  variam dentro do conjunto de vértices do grafo.

Apesar da análise de modularidade ser feita para grafos com peso, os grafos utilizados ao longo deste trabalho foram grafos com arestas sem pesos, o que seria uma ideia similar a um grafo com pesos em que todas as arestas possuem o mesmo peso. Esse detalhe não impacta a detecção de comunidades nos grafos, existindo, inclusive, a opção de configurar no *Gephi* arestas sem peso.

#### Algoritmo

Inicialmente, todos os nós pertencem às suas próprias comunidades, isto é, existem  $N$  comunidades, sendo  $N$  o número de nós, com um nó em cada. O algoritmo é baseado em dois passos que são iterativamente repetidos.

1. O processo analisa todos os nós de maneira ordenada, isto é, a partir do nó 1 até o nó  $N$ . Cada vértice, por sua vez, olha para os seus vizinhos e passa a fazer parte da comunidade de seu vizinho se tiver um aumento no valor da modularidade até então calculada para a comunidade que antes ele participava. Este passo é repetido até que um valor de máximo local para a modularidade seja atingido, de modo que cada nó pode ser considerado várias vezes.
2. Uma vez que o valor local máximo para a modularidade tiver sido atingido, o processo cria um novo grafo em que os nós representam as comunidades formadas. O peso das arestas entre as comunidades é o peso total das arestas entre os nós pertencentes a cada comunidade. Geralmente, o número de nós diminui drasticamente nesta etapa, o que garante a rapidez do algoritmo para grandes redes.

Os passos 1 e 2 são repetidos iterativamente, conduzindo assim a uma decomposição hierárquica da rede.

### 5.1.2 Resultados

A função que calcula a modularidade e separa o grafo em comunidades no software utilizado recebe um parâmetro, chamado resolução, que regula a quantidade de comunidades distintas geradas. Por padrão, vale 1.0, porém pode ser alterado. Valores menores para resolução geram mais comunidades menores, enquanto valores maiores geram menos comunidades maiores, isto é, com mais nós inseridos na mesma comunidade.

Esta subseção irá apresentar resultados de algumas execuções deste algoritmo feitas em redes utilizadas ao longo deste trabalho, após terem sofrido o ataque, isto é, após a criação dos usuários atacantes. O objetivo será analisar a capacidade do algoritmo apresentado neste capítulo de agrupar os atacantes, sem qualquer informação sobre estes nós.

Para cada análise, dois parâmetros serão importantes: resolução, que definirá o número e tamanho das comunidades, e número de nós atacados, uma vez que influencia diretamente na quantidade de arestas que o grafo anonimizado possui, o que é usado no cálculo da modularidade. Para cada rede e caso de quantidade de nós atacados, o algoritmo de criação dos atacantes e suas arestas foi executado apenas uma vez, sendo o grafo gerado analisado.

Para cada análise, os resultados serão expostos em uma tabela. Para cada valor de resolução, o número de comunidades encontradas é apresentado. Além disso, para a comunidade com a maior quantidade de nós atacantes, são mostrados quatro valores: a acurácia, isto é, de todos os nós atacantes, quantos estão nesta comunidade; a precisão, de todos os nós nesta comunidade, quantos são atacantes; a métrica  $F1$ , que é a média geométrica entre a acurácia e a precisão; quantos nós estão na comunidade em questão,  $n$ .

#### Rede do Facebook

Relembrando, a rede inicial é conexa e possui 4039 nós e 88234 arestas. Consequentemente, a rede após o ataque também será conexa, uma vez que o subgrafo formado pelos nós atacantes é conexo - toda aresta  $(a_i, a_{i+1})$  existe e pelo menos um atacante está conectado a um nó vítima. O grafo após ter sofrido o ataque possui  $4039 + 2 * \lceil \log_2(4039) \rceil$  nós, isto é, os nós da rede original mais os 24 nós atacantes, resultando em 4063 nós, independentemente do número de vítimas. O número de arestas, entretanto, irá variar com o número de nós que estão sendo atacados.

Para este estudo, foi utilizado um conjunto de nós vítimas de tamanho igual a 10. Os resultados, variando o valor da resolução utilizado, podem ser encontrados na tabela 5.1. Analisando os resultados apresentados, é possível observar que o algoritmo é capaz de agrupar os nós atacantes em um único grupo, tendo dificuldade, apenas,

em não adicionar à comunidade de atacantes nós não atacantes sem que isso aumente muito o número de comunidades.

Tabela 5.1: Comunidades na rede do Facebook.

<b>Resolução</b>	<b>NºComunidades</b>	<b>Acurácia</b>	<b>Precisão</b>	<b>F1</b>	<b>n</b>
0.05	286	100%	68.57%	82.80%	35
0.1	143	100%	57.14%	75.60%	42
0.5	20	100%	16%	40%	150
1.0	15	100%	20.17%	44.91%	233
2.0	11	95.83%	7.74%	27.23%	297

### Rede Gnutella P2P

A análise dos resultados de detecção de comunidades também foi feita na rede Gnutella P2P, detalhada no capítulo 4, após ter sofrido um ataque de quebra de privacidade em 100 dos seus nós. Esta rede possui 6301 nós e, consequentemente, 26 atacantes, de modo que a rede analisada tenha 6327 nós, uma vez que a análise é feita na rede após a inserção dos atacantes.

Na tabela 5.2 é possível observar alguns resultados diferentes da execução do algoritmo de detecção de comunidades. Ao aumentar o valor da resolução, o algoritmo coloca todos os nós atacantes em uma mesma comunidade, porém com muitos outros nós não atacantes. Ao diminuir a resolução, os nós atacantes começam a não ficar mais em uma mesma comunidade, uma vez que o número de comunidades aumenta, também não conseguindo separar atacantes de não atacantes.

Tabela 5.2: Comunidades na rede Gnutella P2P.

<b>Resolução</b>	<b>NºComunidades</b>	<b>Acurácia</b>	<b>Precisão</b>	<b>F1</b>	<b>n</b>
0.5	68	92.31%	12.43%	33.87%	193
0.8	32	96.15%	5.63%	23.26%	444
1.0	22	100%	3.29%	18.14%	791
2.0	12	100%	0.51%	7.14%	5092

## Modelo Erdős-Rényi

Para o estudo da detecção de comunidades no modelo de rede aleatório de Erdős-Rényi foram utilizados 100 nós, probabilidade de existir uma aresta entre dois nós igual a 0.01 e 30 nós sendo atacados. No caso, são criados 14 nós atacantes.

A partir dos resultados apresentados na tabela 5.3 é possível observar que para valores baixos de resolução, menores iguais a 2.0, o algoritmo não é capaz de agrupar mais de 60% dos nós atacantes em uma mesma comunidade. No caso em que resolução é igual a 3.0, o algoritmo não consegue separar em mais de uma comunidade, agrupando todos os nós do grafo. Para resolução igual a 2.75, o algoritmo separou o grafo em duas comunidades, uma contendo todos os nós atacantes, porém com mais 63 outros nós não atacantes, deixando a outra comunidade com apenas 37 nós não atacantes.

Tabela 5.3: Comunidades na rede do modelo Erdős-Rényi.

<b>Resolução</b>	<b>Nº Comunidades</b>	<b>Acurácia</b>	<b>Precisão</b>	<b>F1</b>	<b>n</b>
0.5	12	14.28%	22.22%	17.81%	9
0.75	8	21.42%	25%	23.14%	12
1.0	5	35.71%	17.86%	25.25%	28
2.0	3	57.14%	18.60%	43.13%	43
2.75	2	100%	18.18%	42.64%	77
3.0	1	100%	12.28%	35.04%	114

# Capítulo 6

## Conclusão e trabalhos futuros

### 6.1 Conclusão

O objetivo principal deste trabalho é analisar o desempenho de um método desenvolvido que fosse capaz de identificar nós determinados de um grafo anonimizado, no intuito de revelar relações entre eles, independentemente da quantidade de nós escolhidos para sofrerem essa violação de privacidade. No capítulo 4 foram apresentados os resultados de mais de 20 mil execuções do algoritmo, em diferentes redes, reais e aleatórias. Em todos os casos, mais de 99% das execuções foram bem sucedidas.

Todas as falhas que ocorreram foram causadas por isomorfismos, diferentemente do trabalho [2], que prova que com alta probabilidade isomorfismos não ocorrem, porém nada diz sobre a existência de automorfismos, que não foram encontrados neste trabalho. O teorema para provar que a probabilidade de um isomorfismo acontecer após a criação do ataque no trabalho [2] tende a zero requer que exista um limite superior para o tamanho do conjunto de nós vítimas, impossibilitando atacar o grafo inteiro, como foi feito neste projeto. Dessa forma, foi preciso deixar o teorema de fora deste trabalho de modo a analisar os resultados de não existirem limites para a quantidade de nós vítimas e o desempenho do algoritmo foi satisfatório.

Conforme falado no capítulo 2, o trabalho proposto em [5] afirma que grafos aleatórios esparsos têm baixo risco de sofrerem um ataque de identificação estrutural, como o proposto neste trabalho, bem sucedido. Entretanto, o nosso trabalho não tenta fazer a identificação dos nós no grafo original, mas apenas do subgrafo de atacantes criado, que é aleatório e denso, de modo que não foram encontrados problemas.

Além disso, por se tratar de um ataque ativo, isto é, um ataque contra a privacidade dos usuários que influencia diretamente na estrutura da rede a ser atacada, o trabalho teve a preocupação de analisar se o ataque seria facilmente detectado. Para isso, foram analisados os resultados de um algoritmo conhecido de detecção

de comunidades para ver se ele seria capaz de separar o subgrafo de atacantes do subgrafo com os nós originais. Para todas as execuções feitas, o algoritmo não foi bem sucedido em criar apenas esses dois grupos e teve dificuldades em gerar comunidades formadas apenas com todos os nós atacantes. Entretanto, em muitas configurações o valor da métrica  $F1$  (média geométrica entre a acurácia e a precisão) foi relativamente alto.

Dessa forma, os resultados do ataque são satisfatórios, uma vez que ele é capaz de identificar os nós determinados no grafo anonimizado na maior parte das vezes e não é facilmente detectado pelo algoritmo de detecção utilizado.

## **6.2 Trabalhos futuros**

### **6.2.1 Maior abrangência**

O algoritmo deve ser avaliado para mais redes, de forma a tornar sólidas as conclusões feitas neste trabalho. Também deve ser avaliado para redes maiores, de forma a determinar se a medida que a rede cresce o ataque tem a mesma eficácia que para redes menores. Além disso, cabe executar para outros conjuntos de nós a serem atacados.

### **6.2.2 Comparações**

Este trabalho tem como objetivo apresentar o método desenvolvido e seu desempenho, porém sem fazer grandes comparações com outros métodos propostos de ataques ativos de quebra de privacidade em redes anonimizadas. Sendo assim, um trabalho futuro seria procurar por mais algoritmos que tenham um objetivo final similar ao descrito ao decorrer deste trabalho e comparar o desempenho destes trabalhos com o descrito nestes estudos.

### **6.2.3 Grafos direcionados**

Este trabalho faz um estudo extenso de ataques contra a privacidade de usuários em grafos não direcionados, porém não analisa o desempenho em grafos direcionados. Uma adaptação no algoritmo, para poder trabalhar com esses grafos, e novas execuções são também interessantes.

### **6.2.4 Remoção de arestas**

É muito comum ter arestas removidas aleatoriamente nas redes que são disponibilizadas publicamente e anonimizadas. Nestes casos, de modo a não impactar fortemente



a estrutura da rede, uma porcentagem específica de arestas costuma ser apagada, sendo essa informação também conhecida, para que quem estiver realizando estudos sobre a rede possa fazer aproximações sobre os dados para obter valores interessantes e realistas.

Sendo assim, é possível pensar em adaptar este projeto para considerar que uma porcentagem das arestas do grafo foram removidas. Isto é, antes da divulgação da rede, após o processo de criação dos atacantes, de modo que informações criadas pelo promotor do ataque possam ser também apagadas. Dessa forma, não é mais possível confiar fortemente nos valores de grau e sequência de grau dos vizinhos atacantes, que são usados para identificar os atacantes no grafo anonimizado.

### **6.2.5 Detecção do ataque**

Este trabalho teve como objetivo principal desenvolver um algoritmo de ataque de quebra de privacidade em grafos anonimizados e analisar seu desempenho. No capítulo 5 foi apresentado um algoritmo específico para analisar se seria possível detectar o ataque, baseado na detecção de comunidade em redes, porém uma análise mais extensa não foi feita. Para próximos trabalhos, seria interessante analisar com outros métodos de detecção de ataques se o algoritmo proposto seria detectado ou não, de modo a adaptá-lo, caso necessário, para evitar sua detecção.

# Referências Bibliográficas

- [1] WIKIPEDIA - THE FREE ENCYCLOPEDIA. “Netflix Prize”. . [https://en.wikipedia.org/wiki/Netflix\\_Prize](https://en.wikipedia.org/wiki/Netflix_Prize), jul. 2017. Acessado em julho de 2017.
- [2] BACKSTROM, LARS; KLEINBERG, JON; DWORK, CYNTHIA;. “Wherefore Art Thou R3579X? Anonymized Social Networks, Hidden Patterns, and Structural Steganography”. . [https://utd.edu/~mxk055100/courses/privacy08f\\_files/social-network-privacy-backstrom.pdf](https://utd.edu/~mxk055100/courses/privacy08f_files/social-network-privacy-backstrom.pdf), mai. 2007. Acessado em julho de 2017.
- [3] WIKIPEDIA - THE FREE ENCYCLOPEDIA. “Graph Isomorphism”. . [https://en.wikipedia.org/wiki/Graph\\_isomorphism](https://en.wikipedia.org/wiki/Graph_isomorphism), jul. 2017. Acessado em julho de 2017.
- [4] WIKIPEDIA - THE FREE ENCYCLOPEDIA. “Graph Automorphism”. . [https://en.wikipedia.org/wiki/Graph\\_automorphism](https://en.wikipedia.org/wiki/Graph_automorphism), jul. 2017. Acessado em julho de 2017.
- [5] HAY, MICHAEL; MIKLAU, GEROME; JENSEN, DAVID; TOWSLEY, DON; LIN, CHAO. “Resisting Structural Re-identification in Anonymized Social Networks”. . <http://www.vldb.org/pvldb/1/1453873.pdf>, ago. 2008. Acessado em julho de 2017.
- [6] CORNEIL, D. G., GOTLIEB, C. C. “An Efficient Algorithm for Graph Isomorphism”, *J. ACM*, v. 17, n. 1, pp. 51–64, jan. 1970. ISSN: 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321556.321562>.
- [7] ERDŐS, PAUL; RÉNYI, ALFRÉD. “On Random Graphs”. . [http://www.renyi.hu/~p\\_erdos/1959-11.pdf](http://www.renyi.hu/~p_erdos/1959-11.pdf), nov. 1958. Acessado em julho de 2017.
- [8] BARABÁSI, ALBERT-LÁSZLÓ; ALBERT, RÉKA. “Emergence of scaling in random networks”. . *Science* 286, 509-512. Acessado em julho de 2017.
- [9] LESKOVEC, J., KREVL, A. “SNAP Datasets: Stanford Large Network Dataset Collection”. . <http://snap.stanford.edu/data>, jun. 2014.

- [10] BARABÁSI, ALBERT-LÁSZLÓ. *Network Science*. Acessado em julho de 2017.
- [11] BLONDEL, VINCENT D.;GUILLAUME, J.-L. R. L. E. “Fast unfolding of communities in large networks”, *Journal of Statistical Mechanics: Theory and Experiment*, v. 2008, n. 10, pp. P10008, 2008. Disponível em: <http://stacks.iop.org/1742-5468/2008/i=10/a=P10008>.