Universidade Federal
do Rio de Janeiro
Escola Politécnica

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.

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, 35 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

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

Referências Bibliográficas: p. 35 – 35.

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

Agradecimentos

Agradeço aos meus pais, os engenheiros que sempre me incentivaram e forneceram todo o suporte para que eu pudesse estar aqui hoje.

Agradeço a todos os professores do curso, 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 dar aula e 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

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

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 ANONYMISED NETWORKS

Pamela Tabak

July/2017

Advisor: Daniel Ratton Figueiredo

Course: Computer and Information Engineering

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

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os traços digitais da interação social humana são facilmente encontrados ao redor da Web. Sites com relações sociais entre seus usuários têm se tornado cada vez mais comuns e facilmente aceitos e usados pela população, para diversos fins. Consequentemente, essas fontes se tornaram, nos últimos anos, dados importantes para estudos de larga escala de redes sociais.

Em muitas dessas fontes de interação social, a interação entre os usuários é pública, isto é, a informação que o usuário x está associado ao usuário y pode ser vista por qualquer usuário. Esse seria o caso do Facebook, por exemplo, em que as relações entre usuários podem ser vistas por outros usuários. Entretanto, existem muitas redes sociais em que essas relações não são disponibilizadas publicamente, de modo que cada usuário conheça apenas as relações que ele possui, desconhecendo a existência de relações entre dois outros usuários da rede. Dessa forma, cada usuário é capaz de estimar corretamente 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 pelo menos uma relação entre o vizinho e outro usuário, no caso ele mesmo, é conhecida. Um exemplo desse tipo de rede seria a de troca de e-mail: cada usuário sabe para quem ele já enviou e de quem ele já recebeu e-mails, porém ele não é capaz de saber se um usuário qualquer já enviou ou recebeu algum e-mail de outro usuário.

Apesar de muitas redes serem privadas, isto é, não disponibiliza a relação entre seus usuários, é comum que as donas dessas redes divulguem o grafo anonimizado que as representam, de modo a permitir que estudos empíricos sejam feitos, que não necessitem da identidade de cada usuário. Um exemplo desse caso foi o da Netflix, descrito em [1], que criou uma competição online para a criação de um sistema de recomendação que a empresa pudesse usar e fosse melhor do que o que ela já usava. Dessa forma, milhares de pessoas ao redor do mundo puderam dar ideias para a criação desse sistema de recomendação, a partir de um grafo anonimizado, em que a identidade dos usuários e filmes foi escondida.

Nesse contexto de divulgação de redes anonimizadas, ataques de quebra de privacidade podem ser desenvolvidos, de modo a revelar relações entre os usuários da rede.

1.1 Tipos de Ataque

Nessa conjectura, existem algumas abordagens proeminentes [2]: ativo, passivo e híbrido.

1.1.1 Ativo

Ataques ativos são algoritmos em que o atacante escolhe um número arbitrário de usuários que ele deseja violar a privacidade, cria um número pequeno de novos usuários, chamados de atacantes, com arestas com os nós atacados e cria um padrão de arestas entre esses novos usuários de modo a criar um subgrafo de novos nós que se destaque na rede anonimizada. Nesse contexto, nós atacados são os usuários da rede que o atacante deseja violar a privacidade.

Dessa forma, o promovedor do ataque passa a ter alguma informação sobre a estrutura da rede, isto é, ele tem conhecimento sobre o grau dos usuários criados e das relações que ele adicionou. A ideia é, a partir dessas informações, encontrar os nós adicionados, nós atacantes, no grafo anonimizado, de modo a identificar os nós atacados a partir da maneira como as relações foram criadas entre esses e os atacantes.

Neste ataque, qualquer usuário da rede pode ser escolhido para ser atacado. Contudo, como ele altera a estrutura da rede, adicionando um novo subgrafo à rede inicial com características próprias, a detecção deste ataque pode ser feita. Isto é, o subgrafo de atacantes é adicionado sem qualquer conhecimento prévio sobre a rede ou suas métricas, de modo que se os administradores da rede realizarem checagens periódicas na mesma, podem, por exemplo, detectar uma mudança na distribuição de grau dos nós e, consequentemente, suspeitar de um possível ataque.

1.1.2 Passivo

Neste ataque, os atacantes são alguns dos próprios usuários do sistema a ser atacado. Eles não criam novos usuários ou novas arestas, usando a informação que eles possuem da rede, deles mesmos, para encontrá-los na rede anonimizada e, a partir disto, tentar descobrir arestas entre usuários a quem eles estão relacionados.

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 não possa ser detectado como um possível ataque.

1.1.3 Híbrido

O ataque híbrido, também conhecido por ataque semi-passivo, surgiu como uma forma de tentar mesclar as boas características dos ataques ativo e passivo, de forma a tentar diminuir, também, os efeitos ruins deles.

Neste ataque, os atacantes também são usuários do sistema. Eles não criam nenhum novo usuário, porém eles criam algumas novas arestas para os usuários atacados antes da rede ser divulgada, de forma a poder atacar qualquer usuário da rede.

1.2 Motivação

É inegável que as redes sociais se tornaram parte do cotidiano de muita gente. O Facebook, uma 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.

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 divulgada e atinge mais pessoas conforme o número de usuários aumenta. Segundo, é inegável que sentimos uma segurança maior quando várias pessoas conhecidas estão utilizando o serviço. A dúvida que resta tirar é: os dados dessas redes realmente não podem ser comprometidos?

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

Dessa forma, a criação de ataques contra a privacidade em redes se torna um assunto cada vez mais interessante por dois motivos, principalmente: 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, de modo que atraia a atenção de pessoas que desejam conhecer relações, como a confirmação de troca de mensagens entre dois usuários.

1.3 Objetivo

O objetivo deste trabalho é apresentar um método desenvolvido para identificar nós pré-determinados no caso das redes em que não se tem informação sobre as relações

entre os demais nós, como a rede de e-mail citada. Isto é, nas redes trabalhadas, cada usuário conhece apenas as suas relações, sendo o objetivo do trabalho encontrar relações entre os nós escolhidos para terem sua privacidade quebrada, chamados de nós atacados. 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 atacados, isto é, a ideia é tornar possível a identificação de qualquer quantidade de nós na rede, inclusive de todos os nós. Para isso, também é levado em consideração que o número de nós a serem adicionados na rede, por ser um ataque ativo, deverá ser um número pequeno quando comparado ao tamanho da rede. O cálculo de número de nós a serem adicionados, chamados de nós atacantes, será detalhado no capítulo 3.

Além disso, a eficácia do método será analisada, para diferentes conjuntos de nós atacados, de tamanhos distintos. Serão considerados apenas dois possíveis resultados do algoritmo: sucesso, quando todos os nós atacados foram identificados, e falha, no caso contrário. Para isso, diferentes redes serão analisadas no capítulo 4, como o modelo Erdõs-Rényi [3] e redes reais, fazendo um estudo comparativo em relação ao número de nós atacados. De modo a tornar possível a avaliação do método, isto é, averiguar quantas vezes foi sucedido ou não, este trabalho irá analisar redes não anonimizadas, simulando a anonimização após a criação dos atacantes.

A escolha pelo ataque ativo ocorreu de modo a tornar possível o ataque a 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 tentar inserir um subgrafo na rede que se destaque, para que o próprio atacante possa o reconhecer. Este trabalho irá abordar, no capítulo 5, brevemente sobre a detecção do ataque. Os grafos não-direcionados, por sua vez, foram optados devido a maior dificuldade em trabalhar com estes na identificação de nós, entretanto, o algoritmo pode ser facilmente alterado de modo a aceitar grafos direcionados.

1.4 A técnica

A técnica desenvolvida terá dois componentes principais: a criação de nós atacantes e a identificação dos nós atacados, que será feita pela recuperação do subgrafo criado no ataque, isto é, 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, isto é, nós atacantes. Cada atacante terá um identificador, que será $atacante_x$, sendo $0 \leq x \leq (atacantes - 1)$. As arestas entre os atacantes serão criadas de maneiras determinísticas e aleatórias, seguindo regras

definidas pelo atacante, de modo a facilitar a identificação desses atacantes no grafo anonimizado.

Cada nó atacado irá se conectar a um subconjunto de nós atacantes, e cada subconjunto destes precisa ser diferente dos demais, uma vez que se o processo de recuperação for capaz de identificar todos os nós atacantes, a identidade de todos os nós atacados será revelada. Isto é, se for possível identificar os atacantes no grafo anonimizado, será possível identificar, por definição, todos os nós atacados. Como foi o promovedor do ataque quem criou as arestas entre os atacantes e os nós atacados, ele sabe exatamente qual nó atacado está conectado a cada subconjunto de atacantes. No final, basta encontrar um único nó no grafo anonimizada que esteja conectado à aquele exato subconjunto de atacantes, que já foi encontrado.

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 atacados, até atingir seu grau máximo ou até não terem mais nós a se conectar: quando todos os nós do grafo são atacados, não existem nós não atacados para completar o grau dos atacantes. Este processo ocorre para aumentar a aleatoriedade do processo e das relações entre os nós atacantes e o nós da rede atacada.

Para a recuperação do subgrafo criado, o atacante possui apenas a informação relativa aos nós e arestas que ele criou, uma vez que nas redes estudadas a informação estrutural da rede, como grau de cada nó, não é revelada. Este processo irá procurar, a partir da estrutura da rede, os nós atacantes em ordem. Isto é, o primeiro passo do processo é procurar pelo atacante 0, o qual se sabe o grau e a sequência de grau dos vizinhos dele que também são atacantes. Dessa forma, o algoritmo irá fazer uma varredura no grafo em busca de nós com o grau igual ao grau deste atacante 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 identificar todos ou chegar à conclusão que não é possível fazer a identificação, resultando em falha.

A partir da maneira como os nós atacados foram relacionados aos nós atacantes, se for possível identificar todos os nós atacante na rede anonimizada, então será possível identificar todos os nós atacados. Como cada nó atacado está conectado ao a um subconjunto de nós atacantes diferentes e todos os nós atacantes foram identificados na rede, é possível a partir dessas informações identificar os nós atacados.

O algoritmo completo é apresentado detalhadamente no capítulo 3.

1.5 Estruturação do documento

No capítulo 2, veremos alguns trabalhos relacionados, isto é, trabalhos que também fizeram estudam ataques em grafos anonimizados e detecção dos mesmos. O capítulo 3 detalha o algoritmo proposto, bem como a tecnologia envolvida no desenvolvimento

do mesmo. 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 método e possíveis extensões e melhorias em trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Neste capítulo serão descritos alguns trabalhos relacionados. Isto é, trabalhos que também fizeram estudam ataques em grafos anonimizados e detecção dos mesmos.

2.1 Wherefore Art Thou R3579X?

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 Criação dos atacantes

Este processo recebe como entrada o grafo a ser atacado e os nós específicos a serem atacados. O número de nós atacados pode variar entre um e todos os nós do grafo.

Número de atacantes criados

Conforme explicado sucintamente na introdução, cada nó atacado irá se conectar a um subconjunto de nós atacantes diferente, de tamanho maior igual a 2. Cada nó atacado precisa, obrigatoriamente, estar conectado a um subconjunto de diferente de nós atacantes, uma vez que o processo de identificação desses nós na rede anonimizada será feito por meio do reconhecimento dos nós atacantes. Dessa forma, se for possível reconhecer os nós criados de forma intencional na rede, será possível reconhecer todos os nós atacados, uma vez que o atacante sabe exatamente as relações existentes entre esses dois grupos de nós.

O tamanho de cada subconjunto de nós atacantes ser maior igual a dois é outra premissa definida do algoritmo. De modo a tornar o processo mais aleatório e evitar a presença de isomorfismos entre o subgrafo de atacantes e o grafo formado pela rede inicial somada ao subgrafo criado, o processo cria arestas entre nós não atacados, se existirem, e nós atacantes. Para isso, é preciso que os nós não atacados possuam subconjuntos de atacantes diferentes à todos os subconjuntos relacionados aos nós atacados, para não confundir um nó atacado a um não atacado. Dessa forma, os nós não atacados só podem se conectar a um nó atacante, isto é, o subconjunto de nós atacantes possui tamanho igual a 1.

Sendo assim, o número mínimo de subconjuntos de atacantes de tamanho maior igual a 2 necessários é igual ao número de nós atacados. Para calcular o número de nós atacantes, isto é, nós a serem criados, será utilizado o cálculo descrito abaixo. X representa o número de nós atacantes, o valor a ser calculado, e Y o número de nós atacados, valor de entrada do processo, determinado pelo atacante.

Para gerar todos os possíveis subconjuntos a partir de X nós atacantes, basta elevar 2 a X: cada atacante pode estar no subconjunto ou não, sendo duas possibilidades, resultando em 2^X. Como os subconjuntos precisam ter tamanho maior igual a 2, é necessário remover os subconjuntos de tamanho igual a 1 (existem X subconjuntos nesse caso) e o subconjunto com tamanho igual a 0, isto é, aquele que não possui nenhum atacante, e só possui 1 subconjunto que representa este caso.

$$Y = 2^X - 1 - X$$

Como não existem limitantes para o valor de nós atacados, Y, sendo $0 \leq Y \leq n$, sendo n o número de nós no grafo. Para o cálculo de número de nós atacantes, o pior caso será utilizado, isto é, quando o atacante deseja atacar o grafo inteiro, Y = n.

Logo, considerando que o número mínimo de diferentes subconjuntos de atacantes de tamanho maior igual a dois é igual a $2^X - 1 - X$, temos:

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

Sabendo que n e X são números inteiros positivos:

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

$$n < 2^X$$

$$X > \log_2 n$$

O propósito do processo é não ter um número de atacantes grande, quando comparado ao número de nós do grafo. Dessa forma, o objetivo será encontrar um valor para X, inteiro e positivo, que seja $\mathcal{O}(\log n)$, isto é, $\log_2 n$ multiplicado por uma constante c:

$$X \leq c * \log_2 n$$

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

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

$$X < c * X$$

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

Consequentemente: $c > 1$

Neste trabalho, será utilizado c = 2. Logo, o número de nós atacantes, X, será igual a duas vezes o log na base dois do número de nós do grafo, isto é:

$$X = 2 * \log_2 n$$

Logo, serão criados $2 * \log_2 n$ novos usuários, os quais o promovedor do ataque conhece informações estruturais, uma vez que ele quem criou esses nós.

Criação das arestas

Como o objetivo do método é encontrar o subgrafo de atacantes criado, esse subgrafo terá propriedades específicas, de modo a tentar se sobressair, ser diferente, do restante da rede. O algoritmo não conhece o grafo que está atacando, logo o subgrafo gerado terá sempre as mesmas propriedades.

1. Criação das arestas entre os atacantes

   (a) Toda aresta do tipo ($atacante_i$,$atacante_{i+1}$) será criada, isto é, o $atacante_0$ estará ligado ao $atacante_1$, o $atacante_1$ estará ligado ao $atacante_2$ e assim por diante, fechando um ciclo entre todos os atacantes (a aresta ($atacante_0$,$atacante_{X-1}$)) também é criada, sendo X o número de atacantes.

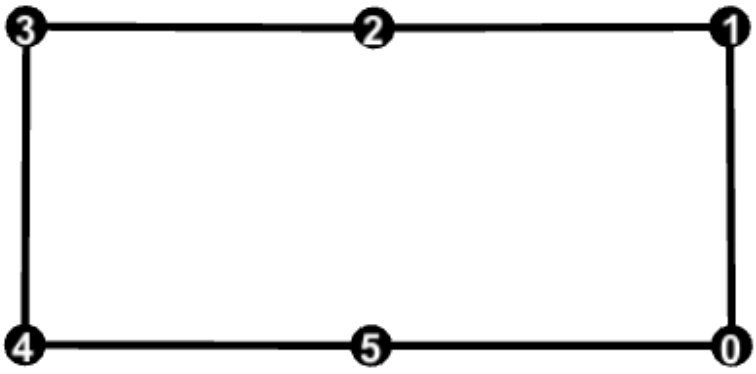

Figura 3.1: Exemplo do subgrafo de atacantes após adição inicial de arestas.

   (b) As demais arestas serão criados aleatoriamente: cada par de nós ($atacante_i$,$atacante_j$), sendo $i < (j + 1)$ terá uma aresta os conectando com probabilidade 1/2.

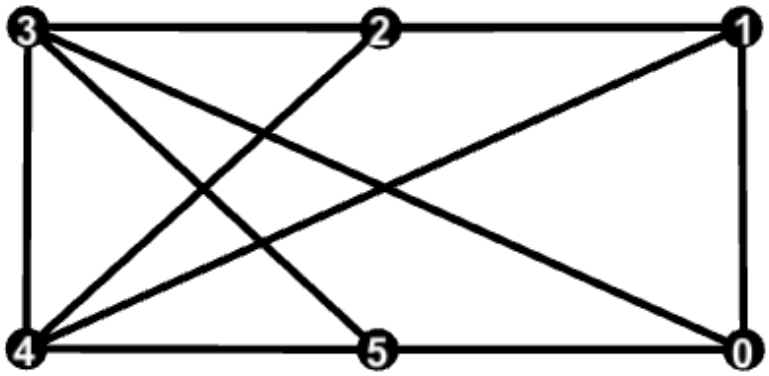

Figura 3.2: Exemplo do subgrafo de atacantes após adição de todas as arestas entre os atacantes.

2. Em seguida, as arestas entre os atacantes e os atacados serão adicionadas. Ao final deste passo, todos os nós atacados precisam estar conectados à um subconjunto de nós atacantes de tamanho maior igual a dois e diferente de todos os demais subconjuntos.

   Para gerar os subconjuntos de atacantes de tamanho maior igual a dois, 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 inicializados com todas as posições contendo o valor 0.

   Para cada nó atacado, o processo verifica se o vetor de marcação possui pelo 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). Em seguida, o vetor é transformado em um número binário de 0's e 1's e é adicionado 1 unidade binário 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 até cada nó atacado

   Dessa forma, é garantido que todo nó atacado estará conectado a pelo menos dois atacantes, de modo que nenhum outro nó atacado esteja conectado à exatamente aos mesmos nós atacantes.

   A tabela 3.1 representa o vetor de marcação ao longo das 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 à um atacante. Por exemplo, a posição 0 do vetor está associada ao $atacante_0$, e assim por diante.

   Para fins demonstrativos, apenas as iterações iniciais são mostradas na tabela.

Para cada nó atacado, o objetivo é encontrar um vetor de marcação com mais de uma posição preenchida com o valor verdadeiro. Logo, o processo é iniciado com o nó atacado 1 e na iteração 1. Nas iterações 1,2 e 3 apenas a soma binária de uma unidade acontece, pois não existem posições suficientes com o valor verdadeiro. Na iteração 4, entretanto, existem, de modo que o nó atacado 1 é ligado, portanto, ao $atacante_2$ e ao $atacante_3$. Em seguida, passamos para o próximo nó atacado, nó atacado 2, e a soma binária acontece. Na iteração 5 apenas a soma acontece e na 6 o nó atacado 2 é conectado aos nós $atacante_1$ e $atacante_3$. O processo continua até todos os nós atacados terem se conectado aos atacantes dessa forma.

Tabela 3.1: Tabela representando vetor de marcação de atacantes a cada instante, 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

3. Por fim, o algoritmo tenta criar arestas entre os atacantes e os nós não atacados. Esse processo é feito de modo a aumentar a aleatoriedade do subgrafo criado e diminuir as chances de encontrar um isomorfismo dele no grafo. Essa parte será discutida mais a fundo mais a frente.

   Neste passo, é importante diferenciar os graus internos e externos do atacante: grau interno é a quantidade de nós atacantes ao qual um nó atacante está relacionado e grau externo é a quantidade de nós não atacantes, isto é, nós da rede inicial, ao qual um nó atacante está relacionado. A soma dessas duas métricas origina o grau de cada nó atacante.

   Para cada nó atacante, um grau externo máximo é gerado a partir de uma distribuição uniforme entre o grau externo que o nó possui até o momento e um número d. Esse número d1 varia de acordo com o grau externo do nó até o momento: se o grau externo for igual a zero, isto é, aquele atacante não está conectado a nenhum nó da rede inicial, então d é igual ao número

de atacantes ($\mathcal{O}(\log n)$). Caso contrário, d segue a seguinte fórmula: $((rand() \pmod X) + 1) * ((rand() \pmod X) + 1) + grauExterno$, em que grauExterno é o grau externo do nó em questão, X é o número de nós atacantes e rand() é a função de c++ que retorna um número aleatório. Essa função aleatório foi inicializada com a semente sendo igual a X.

Uma vez que cada atacante recebe um grau máximo externo maior do que o grau externo que ele possui até o momento, o próximo passo é tentar criar relações entre nós atacantes e não atacados. É necessário, porém, apenas conectar não atacantes a subconjuntos de atacantes diferentes aos subconjuntos gerados para os nós atacados, de modo a não confundir, no processo de reconhecimento no grafo anonimizado, um nó atacante e um não atacante. Para facilitar esse processo e acelerá-lo, uma vez que não será necessário verificar quais subconjuntos já foram usados, o método só associa nós não atacados a subconjuntos de atacantes de tamanho igual a um.

É necessário, aqui, gerar uma lista com todos os nós não atacados. Cada nó dessa lista só poderá ser conectado a um atacante, respeitando a regra definida no parágrafo acima. Assim sendo, para cada nó atacante, enquanto ele não tiver atingido seu grau máximo externo e a lista de nós não atacados tiver tamanho maior que zera, o algoritmo escolhe aleatoriamente um nó não atacado da lista, o removendo da lista, 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 atacados na lista.

Verificação de Automorfismos

Por fim, antes de escrever os arquivos de saída e passar para o próximo passo, o algoritmo faz uma busca simples para verificar a existência de automorfismos no subgrafo de atacantes criado. Não é possível afirmar com certeza, entretanto, se existem automorfismos ou não, pois seria necessário conhecer toda a estrutura da rede formada após os passos acima, porém não é o caso. O atacante só possui a informação de grau dos nós que ele adicionou, então essas informações serão usadas para verificar se existem automorfismos que o algoritmo é capaz de identificar. Se existir, todo o processo descrito acima é refeito. Existe um número de tentativas máximas por execução, de modo a evitar possíveis, porém improváveis dada a descrição do processo, infinitas iterações.

- É necessário, portanto, ter duas definições que serão muito usadas nesta e na próxima subseção:

1. Automorfismo: segundo [4], 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 σ do conjunto de vértices V, tal que para qualquer aresta e = (u,v), σ(e) = (σ(u),σ(v)) é também uma aresta. Ou seja, ele é um isomorfismo de grafos de G para ele mesmo.

   A figura 3.3 representa um grafo que contém automorfismo. A partir da estrutura da rede, nenhum algoritmo é capaz de diferenciar os nós.

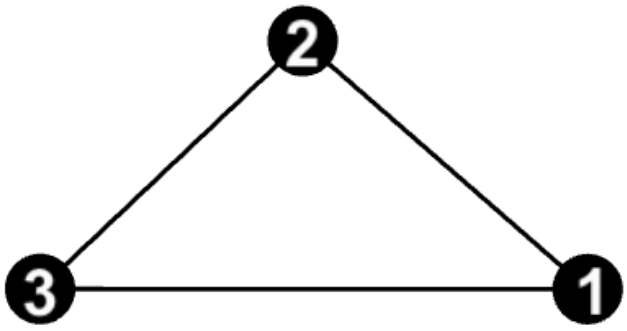

Figura 3.3: Automorfismo em grafo.

2. Isomorfismo: de acordo com [5], um isomorfismo dos grafos G e H é uma bijeção entre os conjuntos de vértices de G e H $f: V(G) \to V(H)$ 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 abaixo representa um exemplo de isomorfismo entre dois grafos.

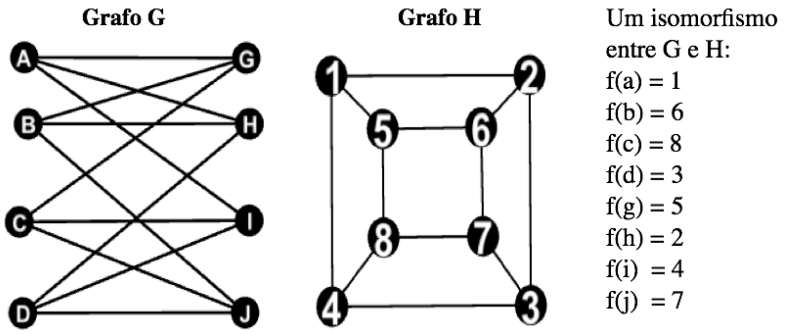

Figura 3.4: Isomorfismo em grafos.

Para o processo final, de identificação dos nós atacados ou reconhecimento do

subgrafo de atacantes, funcionar, o subgrafo criado não pode possuir nenhum automorfismo ou isomorfismo com qualquer outro subgrafo possível no grafo final, resultante do grafo inicial depois de passar pela criação dos nós e das arestas provenientes do ataque.

Não é possível, durante a criação dos atacantes, identificar isomorfismos entre o subgrafo criado e qualquer subgrafo formado a partir da rede completa, uma vez que a estrutura do grafo inicial é desconhecida. Isto é, um isomorfismo entre qualquer subgrafo da rede e o subgrafo de atacantes significaria confundir pelo menos um atacante a um não atacante. Como não temos qualquer informação estrutural sobre os não atacantes, e o reconhecimento de isomorfismos depende dessa informação, não é possível procurar por isomorfismos neste componente.

Pelo mesmo motivo, também não é possível afirmar que existem ou não automorfismos. Porém, podemos 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-completo. O processo criado neste trabalho para procurar por possíveis automorfismos, pelo contrário, possui uma complexidade polinomial e foi inspirada na ideia proposta em [6]: utilizar o grau de cada nó e a sequência de grau dos vizinhos de cada nó.

Desse modo, a ideia consiste em utilizarmos todas as informações que o promovedor 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, para os vizinhos que forem também atacantes, uma vez que não temos informação de grau sobre os demais vizinhos .

Uma condição necessária para existir um automorfismo, a partir dessas informações, é: dois nós atacantes possuírem o mesmo grau, caso contrário eles não poderiam ser confundidos. Outra condição necessária, que depende da primeira ser verdade, é que a sequência de grau dos vizinhos seja igual. Porém, o processo não tem acesso à sequência de grau dos vizinhos completa de cada atacante, ele tem apenas uma parte dela, que são os graus dos vizinhos que são atacantes. Sendo assim, a condição, neste trabalho, é ter dois nós atacantes com o mesmo grau e a sequência de um deles estar contida na do outro. Ambas condições descritas neste parágrafo são necessárias, porém não suficientes.

Para exemplificar a segunda condição descrita, suponha dois nós atacantes, com graus igual a 5. O primeiro nó, $atacante_0$ está conectado a três atacantes e dois atacados, de modo que a sua sequência de grau dos vizinhos possua tamanho três e seja, por exemplo, igual a [1,2,3]. O segundo nó, $atacante_1$, por sua vez, estão conectado a quatro atacantes e um atacado, sendo a sequência de grau dos vizinhos

conhecida dele igual a [1,2,3,4]. Para finalizar, suponha que o grau dos nós atacados conectados à esses dois atacantes seja igual a 4. Dessa forma, as duas sequências de grau dos atacantes fica igual, podendo ocorrer uma confusão entre esses dois nós no grafo anonimizado e, consequentemente, um automorfismo.

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 sequências de graus de vizinhos. Para cada atacante, gera-se a sequência de graus dos seus vizinhos atacantes 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.

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, sendo n o número de nós do grafo inicial. Como $X = 2 * \log_2 n$, 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 saem a partir deste processo e são utilizados nos próximos: o grafo inicial com os novos usuários e arestas, criados por este componente, e um arquivo com as informações estruturais dos atacantes, isto é, o grau de cada atacante e a sequência de graus dos vizinhos dele que são atacantes.

3.1.2 Anonimização do grafo

A proposta deste trabalho é identificar nós pré-determinados, também chamados de nós atacados, 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 atacados 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.

Na ideia original, inicialmente o atacante só conhece todos os nós da rede, adicionando, em seguida, os novos usuários. Em seguida, os detentores da rede anonimizam seus nós, isto é, removem a informação crucial que identificava cada nó, e divulgam a rede assim. Este componente fará exatamente isso, isto é, ele simulará a anonimização da rede a ser divulgada.

Esse processo recebe como entrada a rede gerada pelo processo anterior, isto é, a rede inicial com os nós atacantes adicionados e todas as arestas entre eles e os nós da rede inicial. A saída dele, por sua vez, são dois arquivos: a rede anonimizada, isto é, a mesma rede de entrada, porém com os identificadores anonimizados que foram gerados; o mapeamento de cada nó, ou seja, um mapa entre a identificação que cada nó possuía na rede de entrada e o novo id anônimo gerado. Este arquivo será utilizado para analisar a eficiência do método no final, ao tentar identificar os nós atacados, comparando cada resultado encontrado com o resultado oficial.

- 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:

    1. Cada nó recebe identificador aleatório, que será o índice do seu identificador real na lista com todos os identificadores
    2. Cada nó recebe o identificador de outro nó, isto é, ocorre a permutação entre os identificadores existentes.

    Para exemplificar, a figura 3.5 representa um grafo com 5 nós e arestas não-direcionadas entre eles.

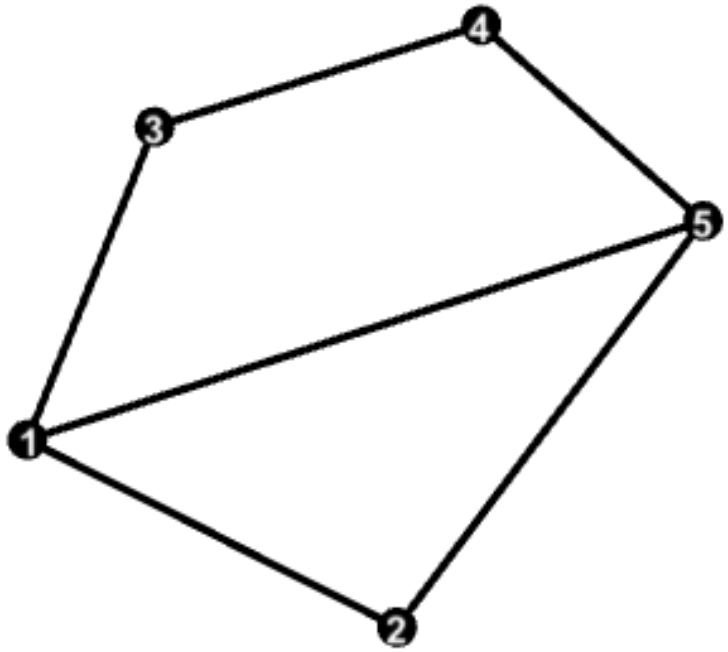

Figura 3.5: Grafo para exemplificar componente de anonimização.

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

Tabela 3.2: Tabela representando 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

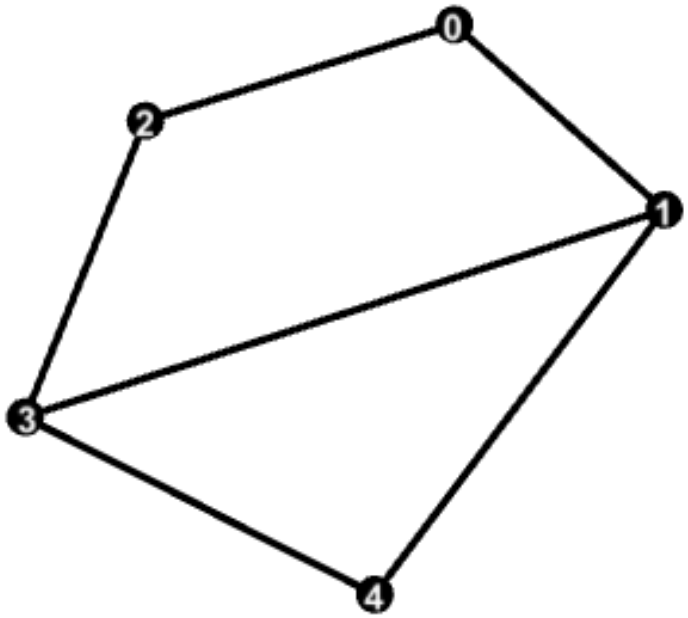

Figura 3.6: Grafo anonimizado pelo processo de anonimização a partir do índice de cada nó.

Tabela 3.3: Tabela representando 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

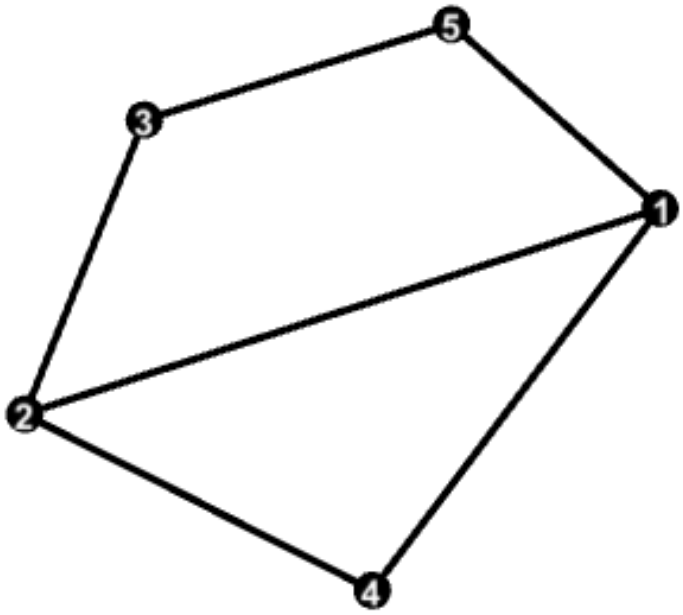

Figura 3.7: Grafo anonimizado pelo processo de anonimização a partir da permutação entre os identificadores.

3.1.3 Recuperação do subgrafo

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

Este componente recebe de entrada os seguintes arquivos, que terão explicados os seus papéis ao longo dessa subseção: o grafo anonimizado após a criação dos

atacantes, isto é, o grafo gerado no processo de anonimização do grafo; o arquivo contendo a informação de cada atacante, isto é, informação de grau e sequência de grau dos vizinhos que são atacantes; o subgrafo de atacantes criado, isto é, todas as relações a partir dos atacantes criados; o arquivo que mapeia o identificador real de cada nó para o identificador anonimizado.

O primeiro passo é carregar todas as informações passadas por arquivos de texto para memória, 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 temos alguma informação estrutural sobre. Para isso, o processo irá procurar em ordem pelos atacantes, isto é, irá procurar pelo $atacante_0$, depois pelo $atacante_1$ e assim por diante.

De modo a encontrar os nós atacantes, o processo irá utilizar uma lista de árvores.

Para encontrar o $atacante_0$, o processo passa por todos os nós do grafo anonimizado e analisa quais nós possuem o mesmo grau que o atacante sendo buscado. Para os que tiverem o mesmo grau, o processo então analisa a sequência de grau dos vizinhos desse nó. Como só temos parte da sequência de grau dos vizinhos do atacante, uma vez que só temos os graus dos vizinhos deste atacante que também são atacantes, é necessário verificar se a sequência de grau dos vizinhos do atacante está contida na sequência de grau dos vizinhos do nó anonimizado que possui o mesmo grau que o $atacante_0$. Se isso acontecer, então o nó anonimizado irá virar uma raiz de uma árvore. Este processo é repetido até verificar todos os nós anonimizados.

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 ser. Por exemplo, o nó na raiz é um possível $atacante_0$.

Dessa forma, para cada atacante (diferente do $atacante_0$), para cada árvore, para cada nó folha, o algoritmo irá buscar por todos os vizinhos do nó folha sendo trabalhado e procurar, utilizando o mesmo procedimento descrito para identificar o $atacante_0$, o próximo atacante. Isto é, se o nó folha está representando um possível $atacante_x$, então o algoritmo irá procurar dentre todos os vizinhos deste nó folha o $atacante_{x+1}$, relembrando que de acordo com a maneira que os atacantes foram criados, todo $atacante_x$ está conectado ao $atacante_{x+1}$.

A figura 3.8 é um exemplo de possível árvore gerada após 3 iterações de busca por atacantes. Existem dois possíveis nós para serem o $atacante_0$, B e G, que possuem o mesmo grau deste atacante e a sequência de grau dos vizinhos atacantes do atacante está contida na sequência de grau dos vizinhos de B e G, existem quatro possíveis nós para o $atacante_1$, (D,A,E,H), e apenas um nó para ser o $atacante_2$.

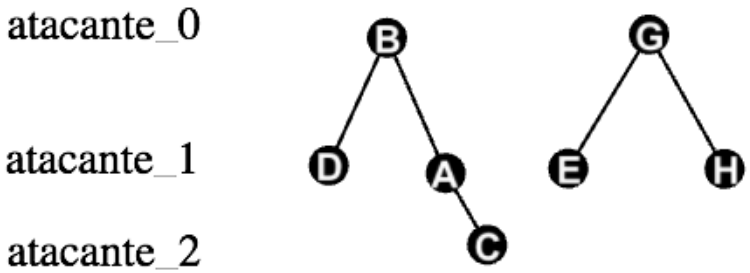

Figura 3.8: Árvore exemplificando a recuperação dos atacantes no grafo anonimizado.

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, sendo X igual ao número de atacantes (considerando que a altura da árvore começa em zero); a soma de nós folhas em todas as árvores ultrapassa um limite estipulado pelo promovedor do atacante. Isto acontece para que o processo não gere milhares e milhares de possíveis subgrafo de atacantes, o que ocorre quando existem muitos possíveis automorfismos ou isomorfismos no grafo anonimizado.

Vale ressaltar que se durante uma iteração um nó folha representando um possível $atacante_x$ não encontrar em seus vizinhos um possível $atacante_{x+1}$, o nó folha em questão é apagado, uma vez que ele não pode representar o $atacante_x$ já que por definição ele está conectado ao $atacante_{x+1}$. Uma vez apagado este nó folha, o algoritmo analisa se o pai deste nó 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 atacante sendo procurado.

Uma vez que o processo tenha terminado, por qualquer dos motivos citados acima, falta 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 acima ele encontrar uma única árvore, de altura igual ao número de atacantes - 1 (relembrando que estamos considerando que a altura inicial da árvore é igual a 0), com número de nós igual ao número de atacantes. Isto é, seria um grafo linha com X nós, sendo X o número de nós atacantes, em que cada nó representa um nó atacante em ordem, isto é, a raiz representa o $atacante_0$, o nó conectado à raiz representa o $atacante_1$ e assim por diante. Se isso acontecer, então o processo funcionou e todos os nós atacados e as relações entre eles podem ser identificadas. Caso contrário, o

processo falhou.

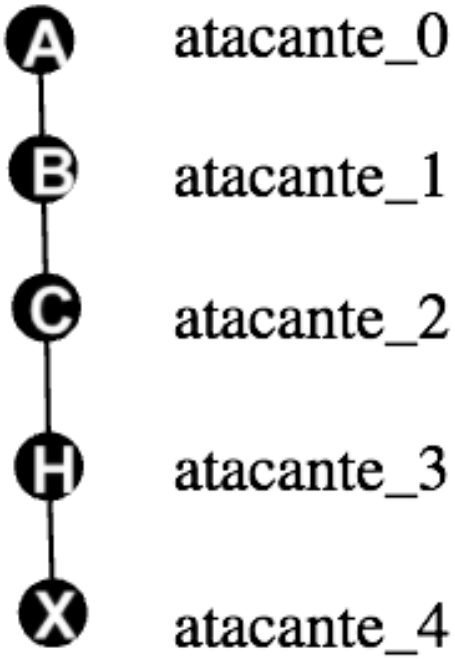

Figura 3.9: Árvore representando o sucesso na recuperação dos atacantes no grafo anonimizado.

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 número de nós igual ao número de atacantes.

- 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 um subgrafo qualquer presente no grafo anonimizado. Dessa forma, um ou mais nós não atacantes são confundidos com nós atacantes.
    2. Automorfismo não identificado do subgrafo, isto é, não é possível diferenciar um par, ou mais de atacantes. Por exemplo, dado um nó qualquer do grafo anonimizado, pelas informações que o processo possui, ele pode ser tanto o $atacante_0$ quanto o $atacante_1$. 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 afirmar que automorfismos não existam.

Para exemplificar esse caso, que o processo de verificação de automorfismos não é capaz de reconhecer inicialmente, vamos supor dois nós de atacantes, com o mesmo grau e o mesmo número de atacantes conectados à ele (mesmo tamanho da sequência de graus dos vizinhos que são atacantes). Porém, as sequências não são iguais: de um é [1,2,3] e de outro é [1,2,4], ou seja, o algoritmo descrito não é capaz de identificar um automorfismo aqui. Agora, por exemplo, vamos supor que o primeiro está conectado a um nó atacado de grau 4 e o segundo está conectado a um nó atacado de grau 3, informações que o ataque não possui. Logo, na busca pelo atacante anonimizado no grafo, existem dois nós com a mesma sequência de grau dos vizinhos [1,2,3,4], podendo caracterizar um automorfismo não identificado pelo processo. Apenas isso não é suficiente para afirmar que existirá um automorfismo, dado que as informações dos nós vizinhos a cada um ele também importam, mas é um indicador de um possível automorfismo.

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. Então se o processo gerar mais automorfismos identificáveis do que o número máximo de vezes, o processo de recuperação do subgrafo irá falhar. Como veremos no próximo capítulo, esse caso é mais difícil de acontecer.

Por fim, o processo utiliza o mapeamento de cada identificador real para o identificador anônimo para verificar o motivo da falha, isto é, verificar se o problema foi um isomorfismo ou um automorfismo.

3.2 Implementação

Os algoritmos descritos neste capítulo foram implementados na linguagem 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, dicionários foram utilizados para o armazenamento eficiente das informações sobre os nós e suas relações. Cada processo utilizada as estruturas que melhor funcionam para si, isto é, que diminuem a complexidade de execução do algoritmo. Por exemplo, se a ideia é fazer muito acesso à uma lista, o *unordered set* possui uma complexidade de busca melhor de

que a de um vetor.

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. 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 facilitar a execução de diferentes grafos. Esses *scripts* executam os três processos descritos acima 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 Erdõs-Rényi [3], em que dois parâmetros são passados: o número de nós do grafo e a probabilidade de existir uma aresta entre quaisquer dois nós.

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, todas as redes utilizadas nesta seção não são anonimizadas, 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 atacados, 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.

Todas as redes apresentadas na seção de resultados, portanto, são redes reais ou redes aleatórias criadas a partir de modelos de redes por este trabalho. Para cada rede e para cada número de nós atacados, que variam entre um e o número de nós na rede, 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 de 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. 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 atacados, o número de nós fica em torno dos milhares.

Os resultados dispostos nesta seção representam a quantidade de sucessos por rede e número de nós atacados, normalizado pelo número de execuções, isto é, quando aparece o valor 1, significa de x execuções o algoritmo acertou todas, por exemplo. No caso de ocorrem falhas, resultados demonstrando o tipo de falha, isomorfismo ou automorfismo, também serão apresentados de maneira similar.

4.2.1 Grafos Aleatórios

Erdõs-Rényi [3]

Um processo para a criação de redes deste modelo foi criado, em que dois parâmetros são passados: o número de nós na rede e a probabilidade de existir uma aresta entre dois nós. Para a execução deste processo, utilizamos 1000 nós para a rede e diferentes valores da probabilidade, representada por p, variando de 0.01 a 0.5. Conforme aumentamos o valor de p, 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 (como p representa um valor probabilístico, ele pode variar entre 0 e 1).

Para cada combinação de p e número de nós atacados, 50 execuções foram feitas, gerando uma nova rede seguindo o modelo aleatório descrito a cada uma delas. Existem 152 combinações diferentes de valores para a probabilidade p de existir uma aresta entre dois nós e número de nós atacados, de modo a serem feitas 7600 execuções deste processo, representados na tabela 4.1. É possível, portanto, reparar que apenas duas execuções não foram sucedidas 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% e outro ao atacar todos os nós da rede com probabilidade de existir uma aresta entre dois nós igual a 50%. Os dois erros foram causados por automorfismos no subgrafo de atacantes.

Tabela 4.1: Tabela representando desempenho do algoritmo no modelo Erdõs-Rényi.

№Atacados	**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

Tabela 4.1 – continuação

№Atacados	p=0.01	p=0.05	p=0.08	p=0.1	p=0.2	p=0.3	p=0.4	0.5
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
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
100	0.98	1	1	1	1	1	1	0.98

Barabasi-Albert (BA) [7]

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, ou seja, 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, todos os vértices estão conectados entre si. O número de nós do clique é consideravelmente menor do que n.

2. A cada iteração, um novo vértice com grau m é adicionado.

3. Os m vértices que terão arestas com este novo vértice, de modo que o seu grau seja igual a m nesta 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.

   Representado o número da iteração pela letra t, é possível descrever a probabilidade de um vértice u, já adiciono 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 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 causa 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*, que possui uma função que recebe os parâmetros m e n definidos pelo usuário e cria grafos não-direcionados respeitando o modelo descrito.

4.2.2 Redes Reais

Facebook

Essa rede foi retirada da base de dados do projeto de análise de redes de Stanford, liderado por Jure Leskovec. Este 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 do Facebook. 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 consideraremos neste trabalho que os dados são reais e faremos a comparação entre os resultados encontrados por este modelo e o dado da rede utilizada.

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, de modo a de fato representar uma rede social, uma vez que possui alto valor de clusterização e baixo valor de diâmetro.

Para esta rede, foram feitas cinquenta execuções para cada número de nós a serem atacados escolhidos, ou seja, 2550 execuções diferentes foram feitas. Cada linha da tabela 4.2 representa a quantidade de sucessos, normalizada pelo número de execuções, para um determinado número de nós atacados.

Tabela 4.2: Tabela representando desempenho do algoritmo em uma rede do Facebook.

№ Atacados	**Desempenho**
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
1100	1
1200	1
1300	1
1400	1
1500	0.98
1600	1
1700	1
1800	1
1900	1
2000	1

Tabela 4.2 – continuação

№Atacados	**Desempenho**
2048	1
2100	1
2200	1
2300	1
2400	1
2500	1
2600	1
2700	x
2800	x
2900	x
3000	1
3100	x
3200	x
3300	x
3400	x
3500	1
3600	x
3700	x
3800	x
3900	x
4000	1
4039	x

Ao todo, o processo foi executado XYZ vezes, em diferentes redes.

Capítulo 5

Detecção do Ataque

Como abordado na introdução, este trabalho se baseia na promoção de um ataque ativo, de modo a criar um subgrafo de novos usuários, chamados de atacantes, de modo a ser possível reconhece-lo, na rede anonimizada, a partir de duas informações estruturais de cada atacante: o grau e a sequência de grau dos vizinhos que são atacantes.

Esta seção irá abordar a probabilidade de um ataque ser identificado. Uma vez que o objetivo deste trabalho é, principalmente, focar no desenvolvimento e desempenho do ataque em si e não tanto em sua detecção, utilizaremos apenas uma técnica, muito conhecida no mundo de redes complexas, chamada de detecção de comunidades, que é detalhadamente explicada no capítulo 9 do [8].

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 enquantos nós pertencentes a diferentes comunidades são fracamente conectados.

Para avaliar a detecção do ataque proposto por esse trabalho, utilizamos o algoritmo de detecção de comunidades disponibilizado no *software Gephi*, uma plataforma de visualização interativa, que foi utilizada para gerar os grafos do capítulo 3, e exploração para todos os tipos de redes e sistemas complexos e dinâmicos.

O programa implementa o método de Louvain [9] 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, que varia 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 $\sum_j A_{ij}$ e m é igual a $\frac{1}{2m}\sum_{i,j} A_{ij}$. Vale ressaltar que apenas os vértices da partição a qual está sendo medida a qualidade são analisados nesta fórmula.

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, igual a 1. 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 é 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

Capítulo 6

Conclusão e Trabalhos Futuros

6.1 Conclusão

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, à medida que a rede cresce o ataque tem a mesma eficácia que para redes menores. Além disso, cabe executar para outros valores de nós atacados e analisar se existem configurações, isto é, determinada rede com determinado número de nós atacados, em que o ataque deixa de funcionar.

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 neste estudo.

6.2.3 Remoção de arestas

É muito comum ter arestas removidas aleatoriamente nas redes que são divulgadas anonimizadas. Nesses 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 estudos empíricos interessantes.

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 da criação dos atacantes, de modo que informações criadas pelo promotor do ataque possam ser também apagadas, como uma aresta entre um nó atacante e um nó atacado. 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.

6.2.4 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 apresentamos uma análise de detecção de comunidade em redes após o ataque ter sido feito, de modo a analisar se esse método seria capaz de detectar a presença dos atacantes e reconhecê-los, 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, 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, mai. 2007. Acessado em julho de 2017.

[3] ERDÕS,PAUL;RÉNYI, ALFRÉD. “On Random Graphs”. . http://www.renyi.hu/~p_erdos/1959-11.pdf, nov. 1958. Acessado em julho de 2017.

[4] WIKIPEDIA - THE FREE ENCYCLOPEDIA. “Graph Isomorphism”. . https://en.wikipedia.org/wiki/Graph_isomorphism, jul. 2017. Acessado em julho de 2017.

[5] WIKIPEDIA - THE FREE ENCYCLOPEDIA. “Graph Automorphism”. . https://en.wikipedia.org/wiki/Graph_automorphism, jul. 2017. Acessado em julho de 2017.

[6] 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.

[7] BARABÁSI, ALBERT-LÁSZLÓ; ALBERT, RÉKA. “Emergence of scaling in random networks”. . Science 286, 509-512. Acessado em julho de 2017.

[8] BARABÁSI, ALBERT-LÁSZLÓ. *Network Science*. Acessado em julho de 2017.

[9] 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¿.