Identificando Ataques Sybil em Redes Sociais Online

Trabalho prático de grafos

Raphael Rodrigeus Campos *

DCC - Universidade Federal de Minas Gerais

Junho 2015

1 O problema

Redes Sociais Online têm se tornado plataformas populares para conduzir uma série de atividades maliciosas, incluindo spam, phishing, malware e falsificação de perfis. Uma Rede Social Online pode ser modelada como um grafo $G = (V, E)$,não direcionado, onde V é o conjunto de vértices, em que cada vértice representa um usuário, e E é o conjunto de arestas, em que cada aresta representa um relacionamento social entre dois usuários da rede [1].

Em uma rede social online, usuários podem criar múltiplas identidades buscando relacionamentos sociais com intenções maliciosas, esse tipo de ataque é conhido como ataque Sybil[1].

Todos mecanismos de defesa existentes tentam isolar vértices Sybil que estão incorporados dentro da topologia de uma rede social. Cada mecanismo determina se um vértice na rede ou é Sybil ou não-Sybil (honesto) da perspectiva de um vértice honesto efetivamente particionando os vértices rede social em duas regiões distintas, a saber: região honesta e a região Sybil. Uma região honesta no grafo G é um subgrafo que consiste de todos os vértices honestos e arestas entre eles. Por sua vez, uma região Sybil, é composta por vértices maliciosos e arestas entre eles. As arestas em G que conectam a região honesta com a região Sybil são chamadas de arestas de ataque.

Diante disso, um mecanismo de defesa Sybil pode ser visto como um algoritmo de particionamento de grafos. Desse modo, o objetivo desse trabalho é apresentar um algoritmo baseado na maximização da condutância normalizada que, dado um grafo de uma rede social, possa particionar o grafo em duas regiões, a região honesta e a região Sybil [1].

1.1 Condutância normalizada

A condutância de um grafo $G = (V, E)$, onde $A \subset V$ e A forma uma comunidade e $B = V - A$, é definida como $C = \frac{e_{AB}}{e_{AA}}$, onde e_{AB} é o número de arestas entre A e B e e_{AA} é o número de arestas dentro de A

A condutância normalizada C_N é definida como:

$$C_N = K - \frac{e_A e_B}{e_A e_A + e_A e_B} \tag{1}$$

onde $e_A = e_{AA} + e_{AB}$ e $e_B = e_{BB} + e_{AB}$ e o valor K de uma comunidade A é definido como:

$$K = \frac{e_{AA}}{e_{AA} + e_{AB}} \tag{2}$$

, assim temos:

$$C_N = K - \frac{(e_{AA} + e_{AB})(e_{BB} + e_{AB})}{(e_{AA} + e_{AB})^2 + (e_{AA} + e_{AB})(e_{BB} + e_{AB})} \tag{3}$$

$$C_N = K - \frac{(e_{BB} + e_{AB})}{(e_{AA} + e_{AB}) + (e_{BB} + e_{AB})} \tag{4}$$

*Matrícula: `2015660911`

Sabemos que $|E|$ é constante e se mantém constante a qualquer mudança de grupo e que $|E| = e_{AA} + e_{AB} + e_{BB} => e_{BB} = |E| - e_{AA} - e_{AB}$, portanto substituindo na equacão(4), temos:

$$C_N = K - \frac{|E| - e_{AA} - e_{AB} + e_{AB}}{e_{AA} + 2e_{AB} + |E| - e_{AA} - e_{AB}} \tag{5}$$

refatorando,

$$C_N = K - \frac{|E| - e_{AA}}{|E| + e_{AB}} \tag{6}$$

logo a condutância normalizada é dada por:

$$C_N = \frac{e_{AA}}{e_{AA} + e_{AB}} - \frac{|E| - e_{AA}}{|E| + e_{AB}} \tag{7}$$

Dessa forma podemos ver mais claramente a relação entre e_{AA} e e_{AB} a medida que escolhemos colocar um vértice de B em A.

2 Algoritmos

O algoritmo 1, *partition*, demonstra como particionar o grafo em dois grupos A e B, onde todos os vértices em A são considerados honestos e em B são considerados Sybil, utilizando a maximização da condutância normalizada.

Seja o grafo $G = (V, E)$ e o conjunto S instâncias de entrada para o algoritmo, onde S é um conjunto de vértices pertencentes a V previamente identificados como Sybil.

Algorithm 1 $partition(G = (V, E), S)$

1: $H \leftarrow V - S$
2: $A \leftarrow get20RandomVerticesFrom(H[1..100])$
3: $B \leftarrow V - A$
4: $beforeC_n \leftarrow G.conductancyNorm()$
5: $(afterC_n, v) \leftarrow \max\{(conductancyNorm(G, v), v) : v \in B\}$
6: **while** $B \neq \emptyset$ **and** $beforeCN < afterCN$ **do**
7: $\quad putVertexInA(G, A, B, v)$
8: $\quad beforeC_n \leftarrow G.conductancyNorm()$
9: $\quad (afterC_n, v) \leftarrow \max\{(conductancyNorm(G, v), v) : v \in B\}$
10: **end while**
11: **return** (A, B)

Na linha 1 do algoritmo é inicializado o conjunto H de honestos, na linha 2 o conjunto A é inicializado com 20 vértices escolhidos aleatoriamente dos primeiros 100 elementos do conjunto H. Na linha 3 B é inicializado com restante dos vértices do grafo, ou seja, inicialmente todos são considerados Sybil exceto os 20 elementos em A. Após as inicializações, na linha 4 é calculado a condutância normalizada atual do grafo G baseado na equação 7. Na linha 5 calcula-se a condutância normalizada simulando a escolha de cada vértice $v \in B$, e assim, é escolhido o vértice que maximiza o valor de C_N.

Na linhas 6, no teste do laço **while** verifica-se se não há elementos em B e portanto não tem mais como particionar, ou então se a condutância normalizada estaguinou ou diminiu.Se qualquer uma das preposições forem verdadeiras, então o laço termina.

No corpo do laço **while** nas linhas 7-9 é inserido o vértice v, que maximiza a condutância normalizada, em A e removido de B (o pseudo código de $putVertexInGroup$ encontra-se no algoritmo 2), e então é recalculado, na linha 8, a condutância do grafo atual e, por fim, na linha 9 é encontrado o vértice $v \in B$ que maximiza a condutância do grafo atual.

Teorema 2.1. *O algoritmo 2, putVertexInA, é executado em tempo $O(|V|)$. E tem complexidade de espaço $O(1)$.*

Demonstração. As operações das linhas 1-2 tem complexidade $O(1)$. As operações das linhas 3-5 tem custo $O(1)$. O laço **for** da linha 6 itera no pior caso $|V| - 1$ vezes, e portanto é $O(|V|)$. Portanto, a complexidade total de tempo do algoritmo é $O(|V|)$. □

Algorithm 2 $putVertexInA(G = (V, E), A, B, u)$

1: $A \leftarrow A \cup \{u\}$
2: $B \leftarrow B - \{u\}$
3: $G.eAA \leftarrow G.eAA + u.outGroupNeighbors$
4: $G.eAB \leftarrow G.eAB + u.inGroupNeighbors - u.outGroupNeighbors$
5: $swap(u.outGroupNeighbors, u.inGroupNeighbors)$
6: **for** $v \in Adj[u]$ **do**
7: **if** $v \in A$ **then**
8: $v.inGroupNeighbors \leftarrow v.inGroupNeighbors + 1$
9: $v.outGroupNeighbors \leftarrow v.outGroupNeighbors - 1$
10: **else**
11: $v.inGroupNeighbors \leftarrow v.inGroupNeighbors - 1$
12: $v.outGroupNeighbors \leftarrow v.outGroupNeighbors + 1$
13: **end if**
14: **end for**

Algorithm 3 $conductancyNorm(G = (V, E), v)$

1: $tmp_{eAA} \leftarrow G.eAA + v.outGroupNeighbors$
2: $tmp_{eAB} \leftarrow G.eAB + v.inGroupNeighbors - v.outGroupNeighbors$
3: **return** $\frac{tmp_{eAA}}{tmp_{eAA}+tmp_{eAB}} - \frac{|E|-tmp_{eAA}}{|E|+tmp_{eAB}}$

Teorema 2.2. *O algoritmo 1, partition, particiona o grafo $G = (V, E)$ em tempo $O(|V|^2 + |E|)$. E tem complexidade de espaço $O(|E|)$, se for levado em consideração o tamanho do grafo, ou $O(|V|)$ caso contrário.*

Demonstração. As operações das linhas 1-3 tem complexida $O(|V|)$. Nas linhas 4 e 5, é possível fazer o cálculo da condutância normalizada em tempo $O(1)$ se o grafo G sempre mantiver atualizado suas váriáveis eAA, eAB. Como mostramos, o calculo da condutância normalizada independe de eBB, e então podemos utilizar o algoritmo 2, $putVertexInA$ para inserir em A os vértices aleatoriamente selecianos por $get20RandomVerticesFrom$. Dessa forma, as variáveis eAA e eAB são mantidas atualizadas, e assim, podemos calcular a condutância em tempo constante. No corpo do laço **while** executamos o algoritmo 2 que tem complexidade $O(|V|)$ no pior caso, pois ele percorre todos os vizinhos do vértice v para atualizá-los. A linha 9 tem complexidade $O(|B|) = O(|V|)$ já que $|B| < |V|$. Como o laço **while** itera no máximo $O(|B|) = O(|V|)$ e para cada iteração percorremos todos os vizinhos do vértice v, que maximiza a condutância normalizada, então percorremos todos as arestas em B, logo é $O(|E|)$. E como executamos a linha 9 $O(|V|)$ vezes temos que o tempo total do algoritmo é $O(|V|^2 + |E|)$. Se o grafo for esparso ou denso a complexidade assintótica é a mesma, que é $O(|V|^2)$. Se não for levado em consideração a representação do grafo, o algoritmo utiliza $O(|V|)$ de espaço para armazenar os vértices dos grupos A e B. Mas se consideramos o grafo, a complexidade varia de acordo com sua representação, portanto, se for utilizado matriz de adjacência tem-se um complexidade de espaço igual a $O(|V|^2)$. Caso seja utilizado lista de adjacência, então a complexidade é $O(|E|)$ onde $|E|$ pode ser $O(|V|)$ se o grafo for esparso, ou $O(|V|^2)$ se for denso. □

3 Métricas

As partições dos grafos foram avaliados utilizandos várias métricas: (a) grau médio dos vértices, (b) modularidade, (c) condutância da região Sybil, (d) codutância da região honesta, (e) coeficiente de agrupamento da região Sybil, (f) coeficente de agrupamento da região honesta, (g) fração de vértices Sybil corretamente classificados, (h) fração de vértices Honestos corretamente classificados, (i) fração de falsos positivos (vértices honestos identificados de forma incorreta como Sybil) e (j) fração de falsos negativos (vértices Sybil identificados de forma incorreta como honestos). As métricas serão descrita com mais detalhes a seguir.

A primeira métrica, grau médio é definido como :

$$GM = \frac{1}{|V|} \sum_{v_i \in V} deg(v_i) \tag{8}$$

onde $deg(v_i)$ é o grau do vértice v_i.

A segunda e terceira métrica são as condutâncias de cada região. A condutância fora introduzida na seção 1. A condutância da região sybil e honesta são dadas, respectivamente, por: $C_{sybil} = e_{AB}/e_{BB}$ e $C_{honesta} = e_{AB}/e_{AA}$.

A quarta e quinta métrica são os coeficientes de agrupamentos de cada região. O coeficiente de agrupamento é apresentado na subseção 3.1.

A sexta métrica, fração de vértices Sybil corretamente classificados, é definida como:

$$fscc = \frac{|B \cap S|}{|S|} \tag{9}$$

onde B é a região classificada como Sybil pelo algoritmo e S é a região Sybil previamente identificada.

A sétima métrica, fração de vértices Honestos corretamente classificados, é definida como:

$$fhcc = \frac{|A \cap H|}{|H|} \tag{10}$$

onde $H = V - S$ e A é a região classificada como honesta pelo algoritmo.

A oitava métrica, fração de falsos positivos, é definida como:

$$fp = 1 - fhcc \tag{11}$$

onde $fhcc$ é a fração de vértices Honestos corretamente classificados.

A nona métrica, fração de falsos negativos, é definida como:

$$fn = 1 - fscc \tag{12}$$

onde $fscc$ é a fração de vértices Sybil corretamente classificados.

As métricas restantes são explicadas nas subseções 3.2 e 3.3

3.1 Coeficiente de agrupamento

Na teoria dos grafos, o coeficiente de agrupamento é uma medida do grau da tendência que os vértices têm de se agruparem.

Existem duas versões dessa medida: global e local. A versão global foi feita para expressar uma indicação geral do agrupamento em uma rede, enquanto a versão local, para expressar uma indicação do quão embutido em uma rede um vértice está.

Abordagem local foi escolhida para cálculo da métrica nesse trabalho.

O coeficiente de agrupamento local de um vértice, em um grafo, quantifica o quão próximo seus vizinhos estão de formarem um clique (subgrafo completo). A vizinhança N de um vértice v_i é definida pelos seus vizinhos imediatamente conectados:

$$N_i = \{v_i : e_{ij} \in E \wedge e_{ji} \in E\} \tag{13}$$

Define-se então k_i como o número de vértices na vizinhança do vértice v_i.

O coeficiente de agrupamento local de um vértice i é dado pela razão entre o número de arestas presentes em sua vizinhança e o número de arestas que poderiam existir no máximo, isto é, o número de arestas que existem no subgrafo completo formado pelos vértices da vizinhança. Para um grafo não direcionado, que é o tratato neste trabalho, o número de arestas do subgrafo completo formado pelos vértices da vizinhança é igual a $\frac{k_i(k_i-1)}{2}$.

Portanto, o coeficiente de agrupamento local de um vértice i é dado por:

$$C_i = \frac{2 \times |\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i\,(k_i - 1)} \tag{14}$$

Como pode ser observado, o coeficiente de agrupamento local é um valor entre 0 e 1. Quando vale 1, todo vizinho de um vértice é também conectado a cada um dos outros vértices dentro da vizinhança; e quando vale 0, nenhum vértice que seja vizinho de um vértice i é conectado a qualquer outro vizinho deste.

O objetivo é calcular o coeficiente de agrupamento médio do grafo. O mesmo é dado pela média dos coeficientes de agrupamento locais de todos os vértices [2]:

$$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{15}$$

Algorithm 4 $clustering\_coefficient(G = (V, E))$

1: $sum \leftarrow 0$
2: **for** all $v_i \in V$ **do**
3: $\quad numEdges \leftarrow 0$
4: $\quad coeff \leftarrow 0$
5: $\quad$ **for** all $v_j \in V$ **do**
6: $\quad\quad$ **if** $i = j$ **then**
7: $\quad\quad\quad$ **continue**
8: $\quad\quad$ **end if**
9: $\quad\quad$ **if** $(i, j) \ni E$ **or** $(j, i) \ni E$ **then**
10: $\quad\quad\quad$ **continue**
11: $\quad\quad$ **end if**
12: $\quad\quad$ **for** all $v_k \in V$ **do**
13: $\quad\quad\quad$ **if** $(j, k) \in E$ **and** $(i, k) \in E$ **and** $(k, i) \in E$ **then**
14: $\quad\quad\quad\quad numEdges \leftarrow numEdges + 1$
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: $\quad coeff \leftarrow \frac{2 \times numEdges}{k_i(k_i - 1)}$
19: $\quad sum \leftarrow sum + coeff$
20: **end for**
21: **return** $\frac{sum}{|V|}$

O algoritmo 4, clustering_coefficient, é uma modificação do algoritmo apresentado em [2]. E calcula o coeficiente de agrupamento médio dado um grafo não direcionado $G = (V, E)$ em tempo $O(|V|^3)$ se for utilizado como representação do grafo matriz de adjacência, pois as operações da linhas 9 e 13 podem ser feitas em tempo $O(1)$. Porém, se for utlizado lista de adjacência o algoritmo terá custo $O(|V|^3 \log |V|)$ se a lista for implementada como uma árvore rubro-negra.

3.2 Modularidade

Modularidade é a fração de arestas que caem em um dado grupo menos a fração esperada de arestas que caem no mesmo grupo. O valor da modularidade encontra-se no intervalo $[-1/2, 1)$. É positivo se o número de arestas dentro dos grupos excedem o número esperado. Para uma dada divisão dos vértices da rede em alguns módulos, a modularidade reflete a concentração das arestas dentro dos módulos comparadas com a distribuição aleatória das conexões entre todos os vértices independentemente dos módulos [6]. A modularidade Q, para um grafo com c comunidades, é defina por:

$$Q = \sum_{i=1}^{c} (e_{ii} - a_i^2) \tag{16}$$

onde e_{ii} é a fração de arestas com vértices na comunidade i:

$$e_{ii} = |\{(u, v) : u \in V_i, v \in V_i, (u, v) \in E\}|/2m \tag{17}$$

e a_i é a fração de arestas com vértices de origem na comunidade i [3][5]:

$$a_i = |\{(u, v) : u \in V_i, (u, v) \in E\}|/2m \tag{18}$$

No nosso caso, após a partição do grafo nós obtemos duas comunidades, A e B, que representam usuários honestos e Sybil respectivamente. Portanto, podemos reescrever a fórmula da modularidade como:

$$Q = (2e_{AA}/2m - ((2e_{AA} + e_{AB})/2m)^2) + (2e_{BB}/2m - ((2e_{BB} + e_{AB})/2m)^2) \tag{19}$$

onde $2e_{AA} = |\{(u, v) : u \in V_A, v \in V_A, (u, v) \in E\}|$ e $2e_{AA} + e_{AB} = |\{(u, v) : u \in V_A, (u, v) \in E\}|$ (análogo para a comunidade B). Portanto, refatorando temos:

$$Q = (e_{AA}/m - ((2e_{AA} + e_{AB})/2m)^2) + (e_{BB}/m - ((2e_{BB} + e_{AB})/2m)^2) \tag{20}$$

Como visto na seção anterior, e_{AA}, e_{AB} e e_{BB} são facilmente obtidos em tempo $O(1)$. Portanto, o cálculo da modularidade do grafo $G = (V, E)$ é $O(1)$.

3.3 Mixing time

Nós definimos o grau de um vértice $v_i \in V$ como o número de vértices adjacentes a v e denotamos-o por $deg(v_i)$. Para G definimos a matriz estocástica de probabilidade de transição P de tamanho $|V| \times |V|$ onde a (i, j)-ésima posição de P é a probabilidade de ir de um vértice v_i para um v_j e é definido como:

$$p_{ij} = \begin{cases} \frac{1}{deg(v_i)} & \text{se } v_i \text{ for adjacente a } v_j \\ 0 & \text{caso contrário} \end{cases} \tag{21}$$

O 'evento' de se mover de um vértice a outro em um grafo é capturado pela Cadeia de Markov que representa um *random walk* no grafo G. Um *random walk* R de tamanho k em G é uma sequência de vértices em G iniciando de um vértice v_i e terminando em v_t, seguindo a probabilidade de transição definida em (21). A cadeida de Markov é dita ergódica se for irredutível e aperiódica. Assim, possui uma única distribuição estacionária π e a distribuição após um *random walk* de tamanho k converge para π quando $k \to \infty$. O *mixing time* T da cadeia de Markov é definido como o tamanho mínimo para o *randam walk* convergir para distribuição estacionária.

Definição 3.1. *(Mixing time) O mixing time (parametrizado por ϵ) de uma cadeia de Markov é definida como*

$$T(\epsilon) = \max_i \min\{t : |\pi - \pi^{(i)} P^t|_1 < \epsilon\} \tag{22}$$

onde π é a distribuição estacionária, $\pi^{(i)}$ é a distribuição inicial concentrada no vértice v_i, P^t é matriz de transição após t passos e $|.|_1$ é distância em variação total. Nós dizemos que a cadeia de Marcov tem um rápido mixing se $T(\epsilon) = poly(\log n, \log \frac{1}{\epsilon})$ onde $n = |V|$. [4]

Teorema 3.1. *(Distribuição estacionária) Para um grafo não direcionado e não ponderado G, a distribuição estacionária da cadeia de Markov sobre G é o vetor de probabilidade $\pi = [\pi_{v_i}]$ onde $\pi_{v_i} = deg(v_i)/2m$. Isto é,*

$$\pi = [deg(v_1)/2m, deg(v_2)/2m, ..., deg(v_n)/2m] \tag{23}$$

onde $m = |E|$ [4].

Teorema 3.2. *(Segundo maior autovalor) Seja P a matriz de transição da cadeia de G com ergodic random walk, e λ_i para $1 < i < n$ autovalores de P. Então todos os λ_i são números reais. Se rotularmos-los em ordem decrescente, $1 = \lambda_1 > \lambda_2 > \lambda_3 > ... > \lambda_n > -1$ é mantido. Definimos segundo maior autovalor μ como $\mu = \max(|\lambda_2|, |\lambda_{n-1}|)$. Então, o mixing time é limitado por:*

$$\frac{\mu}{2(1-\mu)} \log \frac{1}{2\epsilon} \leq T(\epsilon) \leq \frac{\log n + \log \frac{1}{\epsilon}}{1-\mu} \tag{24}$$

[4]

Para calcular o mixing time do grafo G seguimos a definição, iniciando de uma distribuição inicial concentrada em um nodo v_i, e computamos a distribuição após o *random walk* de tamanho t com t grande o suficiente para que a distância $|.|_1$, entre a distribuição após um random walk e a distribuição π, seja menor que ϵ. Repitimos para diferentes pontos iniciais. É uma abordagem possível para ϵ não muito pequeno. O algoritmo 5 descreve esse procedimento.

O algoritmo 5 é executado em tempo $O(T(\epsilon)|V|^3)$, pois a linha 15 é executada em tempo $O(|V|^3)$ se for usado um algoritmo trivial para multiplicação de matriz e tal linha é executada $T(\epsilon)$ vezes. Além disso, o algoritmo tem complexidade de espaço igual a $O(|V|^2)$ que é o mínimo necessário para armazenar as matrizes de transição P e P^t.

O tempo de execução do algoritmo 5 pode ser consideralvemente melhorado se ao invés de andar passo a passo, darmos saltos no espaço de solução fazendo uma espécie de busca binária. O algoritmo 6, *binary_search_mixing_time*, descreve essa solução.

O número de multiplicações de matriz feito pelo algoritmo é dado pela função de recorrência:

$$f(n) = \begin{cases} 2\log(n) + 1 + f(n/2) & \text{se } n > 1 \\ 1 & \text{se } n = 1 \end{cases} \tag{25}$$

Algorithm 5 $mixing\_time(G = (V, E), \epsilon)$

$dist[1..|V|] \leftarrow \infty$
$P[1..|V|][1..|V|] \leftarrow 0$
for all $v_i \in V$ **do**
  $\pi[i] \leftarrow v_i.deg()/2|E|$
  for all $v_j \in Adj[u]$ **do**
    $P[i][j] \leftarrow 1/v_i.deg()$
  end for
end for
$t \leftarrow 0$
$P^t \leftarrow P$
while $\max(dist) > \epsilon$ **do**
  for all $v_i \in V$ **do**
    $dist[i] \leftarrow \frac{1}{2} \sum |P^t(i) - \pi|$
  end for
  $P^t \leftarrow P^t \times P$
  $t \leftarrow t + 1$
end while
return t

Algorithm 6 $binary\_search\_mixing\_time(P_0, P, t, \pi, \epsilon)$

$t_{2steps} \leftarrow 1$
$P_{sqr} \leftarrow P$
$P^t_{before} \leftarrow P^t$
$P^t \leftarrow P_0 \times P_{sqr}$
$D \leftarrow \{d_i : |P^t(i) - \pi|_1 \forall i \in [1, n]\}$
while $\max(D) > \epsilon$ **do**
  $t_{2steps} \leftarrow t_{2steps} \times 2$
  $P_{sqr} \leftarrow P_{sqr} \times P_{sqr}$
  $P^t_{before} \leftarrow P^t$
  $P^t \leftarrow P_0 \times P_{sqr}$
  $D \leftarrow \{d_i : |P^t(i) - \pi|_1 \forall i \in [1, n]\}$
end while
if $t_{2steps} = 1$ **then**
  return t
else
  return $mixing\_time\_DV(P^t_{before}, P, t + \lfloor t_2/2 \rfloor, \pi, \epsilon)$
end if

onde $n = T(\epsilon)$. O pior caso do algoritmo é quando $T(\epsilon)$ é um número potência de 2, ou seja, $T(\epsilon) = 2^x$. Portanto, desenvolvendo a recorrência, assumindo como entrada somente números no formato 2^x, temos:

$$\begin{aligned} &f(n) = 2\log(n) + 1 + f(n/2) \\ &f(n/2) = 2\log(n/2) + 1 + f(n/4) = 2\log(n) - 2\log(2) + 1 + f(n/4) \\ &f(n/4) = 2\log(n/4) + 1 + f(n/8) = 2\log(n) - 2\log(4) + 1 + f(n/8) \\ &f(n/2^{i-1}) = 2\log(n) - 2\log(2^{i-1}) + 1 + f(n/2^i) = 2\log(n) - 2i + 3 + f(n/2^i) \end{aligned} \tag{26}$$

onde $i = \log(n)$. Temos que,

$$\begin{aligned} f(n) &= 1 + \textstyle\sum_{k=1}^{i}(2\log(n) - 2k + 3) \\ f(n) &= 1 + \textstyle\sum_{k=1}^{i} 2\log(n) - \sum_{k=1}^{i} 2k + \sum_{k=1}^{i} 3 \\ f(n) &= 1 + 3i + 2\log(n)\textstyle\sum_{k=1}^{i} 1 - 2\sum_{k=1}^{i} k \\ f(n) &= 2\log(n) \times i + 3i + 1 - 2\textstyle\sum_{k=1}^{i} k \\ f(n) &= 2i^2 + 3i + 1 - 2\frac{i(i+1)}{2} \\ f(n) &= 2i^2 + 3i - i^2 - i + 1 \\ f(n) &= i^2 + 2i + 1 \\ f(n) &= \log^2(n) + 2log(n) + 1 \end{aligned} \tag{27}$$

Pela equação (27) temos que o algoritmo 6 faz $O(\log^2(T(\epsilon))$ multiplicações. Portanto, se utilizarmos uma implementação trivial de multiplicação de matriz temos que o algoritmo é $O(\log^2(T(\epsilon))|V|^3)$. Um melhoramento considerável se comparado com algoritmo anterior.

4 Resultados experimentais e conclusão

Os experimentos foram realizados em um PC - Toshiba Satellite i7 2.1GHz quad-core com 8GB de memória ram. As implementações dos algoritmos apresentados anteriormente são todas sequênciais.

Para os experimentos foram utilizadas as instâncias de grafo inGA (G_A) e inGB (G_B), e instâncias de Sybil previamente classificados sybilGA (S_{GA}) e sybilGB (S_{GB})[1].

O grafo G_A possui 6056 arestas e 768 vértices e o grafo G_B possui 9972 arestas e 1250 vértices. Onde $S_{GA} \subset V_{GA}$ tem tamanho $|S_{GA}| = 256$, e $S_{GB} \subset V_{GB}$ tem tamanho $|S_{GA}| = 250$.

$\|V\|$	$2\|E\|$	partition	mod.	clustering coeff.	mixing time (5)	mixing time (6)
768	12112	0.007923	2e-06	0.045503	167.778	29.3816
1250	19944	0.04821	5e-06	0.293692	468.564	48.7436

Tabela 1: Tempo de execução em segundos para o algoritmo (i) partition; e métricas (ii) modularidade, (iii) coeficiente de agrupamento, (iv) mixing time e (v) mixing time busca binária, ambos mixing time com $\epsilon = 1/|V|$

Na Tabela 1 pode-se observar o tempo de execução do algoritmo de partição e algumas das métricas apresentadas na seção 3. Somente foi mostrado o tempo de execução de algoritmos compuntacionalmente intensivos(exceto a modularidade) e todos esses algoritmos foram executados em um grafo representado por matriz de adjacência. Como pode-se notar, o mixing time leva mais tempo para ser executado, como já era esperado pela análise de complexidade do algoritmo apresentado na Seção 3. É exigido um alto poder computacional para execução das consecutivas multiplicações de matriz na linha 15 do algoritmo 5, e consequentimente, é o algoritmo com maior tempo de execução. Apesar do aumento de velocidade de aproximadamente 10×, o mixing time continuou sendo o algoritmo que leva mais tempo para ser executado.

O tempo de execução mostrado na Tabale 1 para o coeficiente de agrupamento, é a soma do tempo de execução do algoritmo 4 para os grafos induzidos pelos vértices no grupo A (honestos) e grupo B (Sybil). É notório que para o grafo G_B o tempo de execução do coeficiente de agrupamento da um salto. Esse resultado também era esperado, já que o algoritmo apresentado para o cálculo é $O(|V|^3)$.

[1]Os arquivos dessas entradas estão disponíveis em: `http://homepages.dcc.ufmg.br/~rcampos/instances/`

Métricas	G_A	G_B
(a) Grau médio	15.7708	15.9552
(b) Modularidade	0.414632	0.30896
(c) Condutância da região Sybil	0.118043	0.0516529
(d) Condutância da região Honesta	0.179991	0.0126008
(e) Coef. agrup. da região Sybil	0.212556	0.287436
(f) Coef. agrup. da região Honesta	0.438344	0.104614
(g) Fração de sybil corretamente classificados	1	1
(h) Fração de honestos corretamente classificados	0.527344	1
(i) Fração de falso positivos	0.472656	0
(j) Fração de falso negativos	0	0
(k) Mixing time ($\epsilon = 1/\|V\|$)	503	261
(l) μ	0.987612	0.973728

Tabela 2: Métricas para os grafos G_A e G_B após a partição

Na Tabela 2 é apresentado os valores das métricas obtidos após a execução do algoritmo *partition* e fixando o conjunto inicial $V_A = \{5, 7, 10, 13, 18, 20, 23, 35, 37, 45, 49, 53, 59, 67, 79, 82, 90, 92, 96, 99\}$ de vértices no grupo A, para cada par ordenado de entrada (G_A, S_{GA}) e (G_B, S_{GB}). Pode-se observar que a utilização da maximização da condutância normalizada para encontrar usuários Sybil depende muito da topologia do grafo da rede social. Está claro que para o grafo G_B houve 100% acerto na classificação de Sybil e honestos, todavia, o grafo G_A foi obtido uma fração de falso positivos de 0.47 que nos mostra que muitos usuários honestos (quase 50%) foram erroneamente classificados como Sybil.

Métricas	G_A	G_B
(a) Grau médio	15.7708	15.9552
(b) Modularidade	0.436213	0.30896
(c) Condutância da região Sybil	0.0201613	0.0516529
(d) Condutância da região Honesta	0.00992063	0.0126008
(e) Coef. agrup. da região Sybil	0.266577	0.287436
(f) Coef. agrup. da região Honesta	0.314898	0.104614
(g) Fração de sybil corretamente classificados	1	1
(h) Fração de honestos corretamente classificados	1	1
(i) Fração de falso positivos	0	0
(j) Fração de falso negativos	0	0
(k) Mixing time ($\epsilon = 1/\|V\|$)	503	261
(l) μ	0.987612	0.973728

Tabela 3: Métricas para os grafos G_A e G_B com particionamento exato

Na Tabela 3 é apresentados os valores das métricas com particionamento exato do grafo, ou seja, com grupo $A = V - S$ e o grupo $B = S$. Ao compararmos os resultados da Table 2 e 3, podemos notar uma grande mudança no coeficiente de agrupamento da região honesta em G_A,e isso nos dá evidência do porquê o algoritmo, *partition*, não obteve uma classificação tão boa. Na Seção 3 vimos que coefiente de agrupamento quanto mais próximo de 1 maior é o número de clique de tamanho 3 na vizinhaça dos vértices, e portanto, mais conectado o grafo está. Nota-se que o coeficiente de agrupamento médio da região honesta na Tabela 3 é menor que na Tabela 2, ou seja, o algoritmo *partition* coloca em A os vértices honestos mais fortemente conectados com os iniciais V_A. Ainda podemos notar pelos valores das condutâncias que as regiões encontradas pelo algoritmo *partition* não formam tão boas comunidades para grafo G_A quanto poderiam ser se houvesse particionado o grafo corretamente.

Além disso, o mixing time de um grafo está estritamente relacionado com a conectividade do mesmo [4]. Um grafo é fortemente conectado se seu mixing time for pequeno, ou fracamente conectado caso contrário. Podemos notar que G_A tem um mixing time grande e G_B nem tanto, e portanto, podemos concluir que G_B é mais fortemente conectado que G_A.

Sabemos que o grafo G_A não é tão fortemete conectado como G_B e possui um coeficiente de agrupamento na região honesta maior que G_B. Além disso, o grau médio dos grafos evidenciam que G_A é mais denso que G_B. Podemos concluir que o grafo G_A não é particionado

tão bem, pelo fato que os vértices honestos formam uma comunidade que tem subregiões fortemente conectadas. Como a condutância normalizada avalia qualidade de apenas uma comunidade, o algoritmo apresentado falha ao classificar um grafo cujo sua região honesta possui subregiões fortemente conectadas, pois ele tende a deixar no grupo B os vértices que não fazem parte das regiões fortemente conectada.

Referências

[1] F. Benvenuto and A. Magno. Especificação do trabalho prático 2. Maio 2015.

[2] V. H. S. Campos. Abordagem paralela e distribuída para o problema do coeficiente de agrupamento.

[3] P. R. G. Jr and F. M. D. Marquitti. Estrutura e dinâmica de redes ecológicas - modularidade. `http://www.guimaraes.bio.br/A05.pdf`. USP.

[4] A. Mohaisen, A. Yun, and Y. Kim. *Measuring the mixing time of social graphs*. 10th ACM SIGCOMM IMC. ACM, New York, USA, 2010.

[5] U. of Maryland. Modularity. `https://www.cs.umd.edu/class/fall2009/cmsc858l/lecs/Lec10-modularity.pdf`, 2009.

[6] Wikipedia. Modularity (networks). `http://en.wikipedia.org/wiki/Modularity_%28networks%29`.