Um Novo Algoritmo Distribuído para Avaliação e Localização de Centralidade de Rede

Klaus Wehmuth e Artur Ziviani

Laboratório Nacional de Computação Científica (LNCC)
Av, Getúlio Vargas, 333 – 25651-075 – Petrópolis, RJ, Brasil
Email: {klaus,ziviani}@lncc.br

Abstract. *We propose a new distributed algorithm for assessing centrality in complex networks. This algorithm uses only local information at each node, not needing the knowledge of the network's full topology. Besides calculating the centrality value of each node, our proposal also provides a way for locating the most central nodes, also using only local information at each node. We experimentally evaluate the proposed algorithm by comparing it to a recent algorithm and conclude that our algorithm achieves similar results but with a significant reduction on the associated applicability costs. This outcome allows our algorithm to be applied to large-scale networks as we demonstrate by applying our proposal to a real-world large-scale network trace.*

Resumo. *Nós propomos um novo algoritmo distribuído para avaliação de centralidade em redes complexas. Esse algoritmo usa somente informação localizada em cada nó, não necessitando do conhecimento da topologia completa da rede. Além do cálculo do valor de centralidade em cada nó, nossa proposta também provê um meio para a localização dos nós mais centrais, também somente usando informação localizada em cada nó. Nós avaliamos experimentalmente o algoritmo proposto comparando-o a um algoritmo recente e concluímos que nosso algoritmo alcança resultados similares, porém com custo para sua aplicabilidade prática significativamente menor. Esse resultado permite ao nosso algoritmo ser aplicado a redes reais de larga-escala, o que demonstramos aplicando-o a um registro (trace) de uma rede real de roteadores.*

1. Introdução

O conceito de centralidade é uma importante métrica para se avaliar a importância relativa de nós ou arestas individuais na estrutura de uma rede complexa. Devido à sua generalidade e a existência de diferentes formas de interpretar esse conceito, tem-se várias maneiras de definir centralidade de rede [Freeman, 1977; Borgatti, 2005]. O conceito de centralidade pode consequentemente ser utilizado com diversos propósitos para a avaliação de desempenho e comportamento em uma rede complexa, tais como a inferência de informações sobre o nível de conectividade em uma rede, a importância de pessoas e seu papel em uma rede social, a probabilidade de sobrecarga de um determinado nó ou aresta, o impacto em termos de re-roteamento devido à retirada de um nó, entre outras.

Apesar de sua vasta utilidade, o cálculo de centralidade é em geral computacionalmente bastante custoso e requer conhecimento total da topologia, inviabilizando sua aplicação em redes de larga escala, como as muitas encontradas atualmente no cenário

de redes de computadores. Exemplos dessas redes complexas de larga escala atuais incluem diferentes redes interdependentes que compõem a Internet atual, tais como (i) redes centradas em usuários (p.ex., redes sociais online e de microblog); (ii) redes de aplicações (p.ex., redes de email e redes P2P de distribuição de conteúdo); e (iii) redes de infra-estrutura compostas por roteadores. Nesse contexto, há trabalhos propostos visando tornar mais eficiente o cálculo de centralidades tradicionais [Brandes, 2001] ou ainda propor novas formas de centralidade que possam ser obtidas de forma distribuída e que ainda assim possam representar de forma adequada a importância relativa dos diversos nós e arestas da rede [Nanda e Kotz, 2008; Kermarrec et al., 2011].

Neste artigo, nós propomos um algoritmo distribuído para avaliação e localização de centralidade em redes chamado DANCE (*Distributed Assessment of Network CEntrality*). DANCE possui duas contribuições básicas. Como primeira contribuição, DANCE calcula de forma totalmente distribuída valores locais de centralidade de todos os nós de uma rede baseando-se somente na vizinhança de dois saltos de cada nó, sem necessidade de conhecer em qualquer ponto a topologia completa da rede. As demais propostas para avaliação de centralidade de forma distribuída (ver Seção 2) se restringem ao cálculo distribuído dos valores de centralidade, sem fornecer entretanto meios para a localização dos nós de maior centralidade de forma distribuída. Assim, como segunda contribuição, DANCE possui um método para a localização de forma distribuída dos nós de centralidade mais relevante em uma rede – e em particular o nó de maior centralidade de toda rede, o qual nomeamos nó crítico. Essa localização do(s) nó(s) crítico(s) é realizada utilizando-se somente os valores locais de centralidade computados na primeira etapa do DANCE, também sem necessidade do conhecimento da topologia completa da rede.

A nossa proposta DANCE mostra-se simples, eficaz e eficiente. Nós avaliamos o desempenho do DANCE através de sua comparação com outro algoritmo recentemente proposto [Kermarrec et al., 2011] para o mesmo fim, que se baseia no uso de um caminho aleatório para estimação distribuída da centralidade de cada nó da rede. Os resultados experimentais são obtidos em redes sintéticas com diferentes características topológicas. Em termos de efetividade, DANCE alcança resultados similares na avaliação da centralidade ao algoritmo baseado em caminho aleatório, porém com custo para sua aplicabilidade significativamente menor. Isso permite, contrastando com o estado da arte, a aplicação de DANCE para avaliação da centralidade em redes complexas de larga escala, cuja viabilidade é demonstrada pela aplicação do DANCE a um registro (*trace*) referente a uma rede real de larga escala de conectividade de roteadores da Internet.

Este artigo está organizado da seguinte forma. A Seção 2 analisa trabalhos relacionados. Nossa proposta DANCE é apresentada na Seção 3. A Seção 4 discute os resultados experimentais obtidos e a Seção 5 trata das conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Existem várias métricas de centralidade que buscam determinar a importância relativa de um nó em uma rede sob diferentes aspectos, tais como a importância do nó para a conectividade da rede ou para sua capacidade de difusão de informação. Exemplos de métricas de centralidades bem conhecidas são a *betweenness centrality* e centralidade de autovetor. A primeira corresponde em cada nó ao número de caminhos mais curtos entre todos os pares de nós que passam por este nó. A segunda baseia-se no princípio que

conexões de nós com maior grau de conectividade contribuem mais para a centralidade de cada nó que conexões de nós com baixo grau; sendo o PageRank [Brin e Page, 1998] do Google uma variação dessa medida de centralidade.

O cálculo de centralidades tradicionais é tipicamente bastante custoso computacionalmente e requer conhecimento completo da topologia. Nesse contexto, há trabalhos propostos visando tornar mais eficiente o cálculo de centralidades tradicionais [Brandes, 2001]. Dinh et al. [2010] propõem uma nova métrica de centralidade menos custosa computacionalmente, porém ainda requerendo conhecimento completo da topologia.

Alternativamente, como nossa proposta, alguns trabalhos investigam formas descentralizadas de avaliação de centralidade em redes, buscando assim evitar a necessidade de conhecimento completo da topologia. Lehmann e Kaufmann [2003] propõem uma forma decentralizada de cálculo para centralidades baseadas na descoberta e contabilização dos caminhos mínimos entre os nós. Esse trabalho introduz conceitos básicos na obtenção de centralidades de forma distribuída, entretanto o algoritmo proposto em geral tem um custo computacional ainda proibitivamente alto para aplicação em redes de grande porte. Nanda e Kotz [2008] introduzem uma medida de centralidade chamada *Localized Bridging Centrality* (LBC). Essa medida de centralidade é avaliada utilizando apenas os vizinhos diretos de cada nó da rede e dirigida especificamente para localização de pontes (arestas cuja remoção fragmentam a rede), tendo sido aplicada a redes sem fio de pequeno porte apenas.

Recentemente, Kermarrec et al. [2011] apresentam uma nova medida de centralidade chamada *Second Order Centrality* (SOC) que é obtida por meio de um único caminho aleatório perpétuo computando-se o desvio padrão do intervalo de tempo entre visitas consecutivas desse caminho aleatório a cada nó da rede. Esse método distribuído apresenta bons resultados como medida de centralidade, porém o uso de um único caminho aleatório faz com que se torne difícil estabelecer um critério de parada para o cálculo, além da proposta apresentar um alto custo em termos de mensagens e tempo de convergência.

Em comparação com o estado da arte em avaliação distribuída de centralidade, nossa proposta DANCE contribui com uma solução simples cuja efetividade é similar ao SOC [Kermarrec et al., 2011], porém com um custo para sua aplicabilidade significativamente menor (ver Seção 4). Além disso, todas as demais propostas de avaliação distribuída de centralidade se restringem à avaliação da centralidade em cada nó da rede, não possuindo uma solução para a localização de forma distribuída dos nós mais críticos em termos de centralidade, o que constitui a segunda contribuição de nossa proposta.

3. Algoritmo DANCE

Nós propomos um novo algoritmo distribuído chamado DANCE para avaliar a centralidade dos nós de uma rede e localizar os nós mais centrais utilizando apenas informações localizadas de cada nó da rede. A base lógica para isso é a seguinte premissa:

Premissa: se um determinado nó é central para a rede como um todo, ele também será central localmente para uma vizinhança estabelecida em torno desse mesmo nó.

DANCE explora esse conceito simples e intuitivo de forma a determinar a centralidade dos nós da rede de forma distribuída e posteriormente identificar os nós mais centrais. Uma contribuição chave da proposta é justamente mostrar que mesmo baseado nessa premissa simples, pode-se obter uma solução eficaz e eficiente.

3.1. Avaliação distribuída de centralidade

Cada nó da rede considera sua vizinhança de dois saltos, ou seja, a sub-rede formada pelo nó em questão, seus vizinhos diretos e os vizinhos dos seus vizinhos. Essa vizinhança de dois saltos a partir de um determinado nó n será doravante denominada $2\_vizinhança_n$. Após a determinação da $2\_vizinhança_n$ de cada nó n da rede[1], é utilizada uma métrica calculada para a $2\_vizinhança$ de cada nó. O resultado referente a essa métrica é considerado como a medida de centralidade do nó para o qual a $2\_vizinhança$ foi estabelecida. A métrica utilizada pode variar, gerando medidas de centralidade diferentes, e a operação geral do DANCE independe da métrica específica. Um exemplo simples de métrica de centralidade que será utilizada neste trabalho é o volume da $2\_vizinhança$ considerada. Utilizando essa métrica, a medida de centralidade de cada nó corresponde à soma dos graus de cada nó que compõe sua $2\_vizinhança$. O Algoritmo 1 ilustra o mecanismo geral do DANCE para o cálculo da centralidade de cada nó da rede de forma distribuída.

Algoritmo 1 DANCE_DISTRIBUTEDCENTRALITYCOMPUTATION()

```
1: {at each node n in the network}
2: {define shared variable to contain the topology of the 2-hop neighborhood around this node.}
3: define 2_neighborhoodTopology
4: {define shared variable for the centrality value at this node.}
5: define centrality
6: {launch process that sets the shared variable 2_neighborhoodTopology.}
7: fork discover_2_NeighborhoodTopology()
8: {launch process that sets the shared variable centrality.}
9: fork calculateCentrality()
```

3.1.1. Descoberta da topologia da vizinhança a dois saltos de cada nó

O Algoritmo 2 mostra como é realizada no DANCE a descoberta da topologia da sub-rede referente à vizinhança de dois saltos de cada nó da rede. Cada nó envia uma mensagem para cada um de seus vizinhos que por sua vez repetem esta mensagem aos seus vizinhos. Ou seja, o número de mensagens geradas por um nó será igual ao seu grau somado ao grau de cada um de seus vizinhos, de forma que este número de mensagens será também igual ao volume da vizinhança de raio 1 do nó em questão. Assim, a complexidade de mensagens esperada do Algoritmo 2 é $O(n \times d_m^2)$, onde n é o número de nós e d_m é o grau médio da rede. Cada nó pode determinar sua $2\_vizinhança$ de forma independente dos demais nós da rede, logo isso pode ocorrer em paralelo. Portanto, o tempo necessário para isso é limitado ao tempo de propagação de dois saltos. Como consequência, a complexidade de tempo esperada para o Algoritmo 2 é $O(1)$ passos discretos.

3.1.2. Cálculo distribuído da métrica de centralidade

O objetivo do Algoritmo 3 é determinar o valor da centralidade de cada nó baseado na sua $2\_vizinhança$. Em princípio, a métrica deve ser recalculada toda vez que houver uma mudança na topologia da $2\_vizinhança$ percebida pelo nó (ver Algoritmo 2). Como a determinação da $2\_vizinhança$ depende do recebimento das mensagens de topologia local enviada pelos demais membros da $2\_vizinhança$, um temporizador é usado para

[1]Essa determinação pode ser regular ou reativa a mudanças na topologia, possibilitando lidar com dinamismos da rede, uma possibilidade que pretendemos explorar como trabalho futuro.

Algoritmo 2 DISCOVER_2_NEIGHBORHOODTOPOLOGY()

```
1: 2_neighbohoodTopology ← EMPTY
2: localTopologyChanged ← TRUE
3: while TRUE do
4:     waitFor receivedMSG OR localTopologyChanged
5:     if receivedMSG then
6:         if receivedMSG is NEW then
7:             mark receivedMSG as not NEW
8:             update 2_neighbohoodTopology
9:             signal 2_neighbohoodTopology_updated
10:            if receivedMSG.ttl > 0 then
11:                decrement receivedMSG.ttl
12:                send receivedMSG to All Neighbors
13:            end if
14:        end if
15:    end if
16:    if localTopologyChanged then
17:        MSG ← createMSGAllNeighbors(ttl ← 1)
18:        send MSG to All Neighbors
19:        localTopologyChanged ← FALSE
20:    end if
21: end while
```

Algoritmo 3 CALCULATECENTRALITY()

```
1: Timer ← DISABLE
2: while TRUE do
3:     waitFor 2_neighbohoodTopology_updated OR Timer.done
4:     if 2_neighbohoodTopology_updated then
5:         restart Timer
6:     end if
7:     if Timer.done then
8:         calculate metric
9:         if metric has changed then
10:            signal metric_Changed
11:        end if
12:    end if
13: end while
```

limitar avaliações desnecessárias da métrica antes de todas as mensagens poderem ter sido recebidas. O Algoritmo 3 não envolve geração de mensagens e, portanto, não tem custo do ponto de vista de tráfego de mensagens na rede. Dependendo da definição da métrica utilizada para a centralidade, esta poderá ter um maior ou menor custo computacional. Entretanto, o cálculo da métrica utilizada fica restrito à sub-rede determinada pela $2\_vizinhança$ do nó em questão. Isso faz com que em geral o custo computacional seja bastante modesto quando comparado ao custo de avaliar a mesma métrica considerando a rede completa. A determinação precisa da complexidade envolvida depende diretamente da métrica escolhida. Por exemplo, para a métrica de volume da $2\_vizinhança$ mencionada anteriormente, a complexidade computacional é $O(|2\_vizinhança_n|)$, onde $|2\_vizinhança_n|$ é o número de nós existentes na $2\_vizinhança$ centrada no nó n.

3.2. Localização distribuída de nós críticos

Uma vez determinados valores de centralidade para cada nó da rede, um desafio é localizar os nós críticos da rede de forma distribuída. Para isso, primeiramente, introduzimos uma definição para *nós críticos* e *nós semi-críticos*. O processo de determinação de nós críticos é baseado na seleção do nó de maior valor de centralidade de uma $2\_vizinhança$. Nesse processo (apresentado detalhadamente adiante), cada nó de cada $2\_vizinhança$ seleciona o nó de mais alto valor de centralidade de sua $2\_vizinhança$. Para que um nó seja considerado um nó crítico, é necessário que todos os nós de sua $2\_vizinhança$ o tenham selecionado como o nó de maior valor de centralidade. Como isso implica que mesmo os

nós da periferia da $2\_vizinhança$ selecionaram o nó crítico como sendo o nó de maior centralidade visível para eles, ocorre que todo nó crítico é um máximo local de centralidade em um raio de 4 saltos, e em particular que o nó de maior centralidade da rede será necessariamente um nó crítico. Quando um nó seleciona a si próprio como sendo o nó de mais alta centralidade de sua $2\_vizinhança$, ele é considerado um nó semi-crítico. Um nó pode ser semi-crítico sem ser um nó crítico. Esses nós semi-críticos possuem um papel importante na localização de nós críticos conforme será detalhado adiante.

Após determinados os nós críticos, cada nó da rede está associado a um e somente um nó critico. Isso ocorre porque cada nó da rede tem um nó em sua $2\_vizinhança$ que ele considera como de maior centralidade. Considerando essa relação como transitiva, teremos uma sequência de valores crescentes até chegar a um nó crítico. Podemos então usar o conceito de nó crítico associado a cada nó da rede. Esse conceito é utilizado para fazer com que cada nó crítico da rede possa ter conhecimento de todos os demais nós críticos existentes na rede. Para que isso aconteça, cada nó crítico no momento em que descobre seu *status* de crítico, informa aos nós que o selecionaram como crítico (todos os nós de sua $2\_vizinhança$) e cada um desses nós propaga esta informação para os nós que o selecionaram como o nó de maior centralidade. Ao final desse processo, cada nó da rede conhece o nó crítico ao qual está associado.

O Algoritmo 4 apresenta uma visão geral dos passos necessários para localização dos nós críticos da rede. A seguir, apresentamos a parte do algoritmo que permite a cada nó da rede identificar o nó crítico ao qual este está associado e também a parte do algoritmo para que cada nó crítico conheça todos os demais nós críticos de rede.

Algoritmo 4 DANCE_CRITICALNODELOCATION()

```
1: {at each node n in the network}
2: {define shared variable to contain the id of the critical node associated to this node.}
3: define associatedCriticalNode
4: {launch process that identifies the critical node associated to this node.}
5: fork IDENTIFYCRITICALNODE()
6: {launch process that makes critical nodes known to each other.}
7: fork COMPARECRITICALNODE()
```

3.2.1. Identificação de nós críticos

O Algoritmo 5 é responsável pela identificação do nó crítico associado a cada nó da rede. Para tanto, é necessário primeiramente que cada um dos nós críticos e semi-críticos da rede tome conhecimento desse seu *status*, o que é alcançado pelo Algoritmo 6 disparado logo no início (linha 1) do Algoritmo 5. Assim, analisaremos inicialmente o Algoritmo 6 e, em seguida, o restante do Algoritmo 5.

No Algoritmo 6, cada nó envia seu valor de centralidade para os demais nós de sua $2\_vizinhança$. Quando cada nó recebe os valores de centralidade de todos os outros nós de sua $2\_vizinhança$, o mesmo seleciona o nó com o maior valor de centralidade e notifica este nó de que ele é o nó de maior centralidade visível para si. Caso o nó tenha selecionado a si mesmo como o nó de maior centralidade da sua $2\_vizinhança$, ele imediatamente se identifica como um nó semi-crítico. Cada nó então contabiliza o número de seleções recebidas. Se um determinado nó for selecionado por todos os nós de sua $2\_vizinhança$ como sendo o de maior centralidade, este nó é identificado como um nó crítico. Caso haja empate no valor de centralidade, a identidade dos nós é usada

Algoritmo 5 IDENTIFYCRITICALNODE()

```
1: fork CRITICALNODESELECTION()
2: while TRUE do
3:     waitFor nodeCritical OR nodeSemiCritical OR Received criticalNodeIdMSG
4:     if nodeCritical then
5:         criticalNodeIdMSG ← createCriticalNodeIdMSG(thisNode)
6:         send criticalNodeIdMSG to All nodes that selected thisNode
7:     end if
8:     if nodeSemiCritical then
9:         criticalNodeId ← locateAssociatedCriticalNode(thisNode)
10:        criticalNodeIdMSG ← createCriticalNodeIdMSG(criticalNodeId)
11:        send criticalNodeIdMSG to All nodes that selected thisNode
12:    end if
13:    if Received criticalNodeIdMSG then
14:        associatedCriticalNode ← criticalNodeIdMSG.Id
15:        send criticalNodeIdMSG to All nodes that selected thisNode
16:        signal associatedCriticalNodeIdentified
17:    end if
18: end while
```

como critério de desempate, garantindo assim a existência de um único nó crítico em certa região. O Algoritmo 6 necessita que cada um dos nós da rede envie seu valor de centralidade para todos os seus $2\_vizinhos$ e, portanto, é semelhante ao Algoritmo 2 de descoberta da topologia de $2\_vizinhança$ de cada nó. Assim, pela mesma argumentação apresentada na Seção 3.1.1 para o Algoritmo 2, a complexidade de mensagens e complexidade de tempo esperadas para o Algoritmo 6 são $O(n \times d_m^2)$ e $O(1)$, respectivamente, onde n é o número de nós e d_m é o grau médio da rede. Através do Algoritmo 6, cada nó da rede terá ciência se ele mesmo é ou não um nó crítico ou semi-crítico, e necessariamente o nó com a maior centralidade da rede será um nó crítico. Dependendo da topologia da rede, é possível haver mais do que um nó crítico na rede.

Uma vez que cada nó crítico ou semi-crítico esteja ciente de sua condição (Algoritmo 6), podemos analisar o restante do Algoritmo 5, responsável por fazer com que cada nó da rede conheça o nó crítico associado a ele. Quando um nó percebe ser um nó crítico, ele notifica este fato a todos os nós de sua $2\_vizinhança$ que o selecionaram como sendo o nó de maior centralidade visível a estes. Por sua vez, quando um nó recebe a notificação com a identidade do nó crítico, ele armazena a identidade do nó crítico como sendo o nó crítico associado a ele e repassa a mensagem de notificação para todos os nós que o selecionaram como nó de maior centralidade, já que necessariamente o mesmo nó crítico também será o nó crítico associado a estes nós. Assim, a associação de nó crítico se dá de forma transitiva através da seleção de nó com maior centralidade. Quando um nó percebe ser semi-crítico, ele localiza o nó crítico associado a ele utilizando o Algoritmo 7 e propaga esta informação para os nós que o selecionaram, que por sua vez irão também repassá-la. É interessante notar que este tratamento diferenciado dos nós semi-críticos é necessário porque a mensagem propagada a partir do nó crítico não chegará até um nó semi-crítico. Isso decorre do nó semi-crítico por definição selecionar a si próprio e, em consequência, não receber a informação propagada pelo nó crítico.

Para analisar a complexidade de mensagens do Algoritmo 5, devemos notar que cada nó receberá no máximo uma mensagem. Isso ocorre porque os nós críticos e semi-críticos não recebem nenhuma mensagem e todos os demais nós recebem uma única mensagem originada pelo nó que cada um deles selecionou como sendo o de maior centralidade. Como cada mensagem passará por no máximo tantos saltos quanto for o diâmetro da rede, podemos portanto concluir que a complexidade de mensagens desse Algoritmo

Algoritmo 6 CRITICALNODESELECTION()

```
1: Timer ← DISABLE
2: while TRUE do
3:     waitFor metric_Changed OR Received metricMSG OR Timer.done OR Received selectMSG
4:     if metric_Changed then
5:         metricMSG ← createMetricMSG(ttl ← 1)
6:         send metricMSG to All Neighbors
7:     end if
8:     if Received metricMSG then
9:         restart Timer
10:        originNode.metric ← metricMSG.metric
11:        if originNode.metric > thisNode.metric AND metricMSG.ttl > 0 then
12:            decrement metricMSG.ttl
13:            send metricMSG to All Neighbors
14:        end if
15:        if Received metricMSG from All Nodes in 2_neighborhood then
16:            Timer ← DISABLE
17:            selectedNode ← node with highest received metric
18:            if selectedNode ≠ thisNode then
19:                selectMSG ← createSelectionMSG(thisNode)
20:                send selectMSG to selectedNode
21:            else
22:                signal nodeSemiCritical
23:            end if
24:        end if
25:    end if
26:    if Timer.done AND Not Received metricMSG from ALL Nodes then
27:        send requestMSG to all missing Nodes
28:    end if
29:    if Received selectMSG then
30:        Account received selectMSG
31:        if Received selectMSG from All Nodes in 2_neighborhood then
32:            signal nodeCritical
33:        end if
34:    end if
35: end while
```

é $O(n \times Dg)$, onde Dg é o diâmetro da rede. Para determinar a complexidade de tempo, podemos notar que também não ocorrem ciclos na propagação da informação de identidade do nó crítico associado. Dessa forma, o caminho mais longo que pode ser percorrido será o diâmetro da rede. Podemos, portanto, concluir que a complexidade de tempo será $O(Dg)$ passos discretos, onde Dg é o diâmetro da rede.

O Algoritmo 7 objetiva localizar o nó crítico associado a qualquer nó da rede. Partindo de um nó qualquer da rede, consideramos o nó de maior centralidade selecionado por este. Se o nó selecionado for diferente dele mesmo, este nó não é um nó semi-crítico e assim o processo de localização continua recursivamente a partir do nó selecionado pelo nó corrente. Caso o nó corrente seja semi-crítico (ou seja, ele percebe a si próprio

Algoritmo 7 LOCATEASSOCIATEDCRITICALNODE($currentNode$)

```
Input: currentNode
Output: criticalNode
1: waitFor node selected by currentNode Is DEFINED
2: highCNode ← node selected by currentNode
3: if currentNode is NOT semi-critical then
4:     return LOCATEASSOCIATEDCRITICALNODE(highCNode)
5: else
6:     if currentNode is critical node then
7:         return currentNode
8:     else
9:         nextNode ← GETNODEKNOWSHIGHERC(currentNode)
10:        return LOCATEASSOCIATEDCRITICALNODE(nextNode)
11:    end if
12: end if
```

como o nó de maior centralidade em sua $2\_vizinhança$), este nó também pode ser um nó crítico. Se isso ocorrer, o nó crítico foi localizado e o algoritmo retorna sua identidade. Caso contrário, isso significa que pelo menos um nó na $2\_vizinhança$ do nó semi-crítico selecionou um outro nó como o nó de maior centralidade. Assim, esse nó de maior centralidade não está na $2\_vizinhança$ do nó semi-crítico, mas está na $2\_vizinhança$ de um dos nós que participam da $2\_vizinhança$ do nó semi-crítico. Nesse caso, o nó semi-crítico escolhe o nó de maior centralidade que tenha selecionado outro nó que não seja o nó semi-crítico e o algoritmo continua recursivamente a partir deste nó. É importante notar que o caminho percorrido pelo algoritmo é acíclico e totalmente determinístico.

3.2.2. Comparação de nós críticos

No Algoritmo 8, cada nó compara a identidade de seu nó crítico associado com a identidade do nó crítico associado de seus vizinhos. No caso de vizinhos possuírem diferentes nós críticos, o nó corrente envia esta informação ao seu nó crítico associado, que desta forma toma conhecimento da existência de um outro nó crítico na rede. Toda vez que um nó crítico toma conhecimento de um outro nó crítico, ele envia a esse nó crítico a lista de nós críticos que ele já conhecia e notifica esses nós críticos já conhecidos da existência de um outro nó crítico. Considerando a rede conectada, após a convergência do algoritmo cada nó crítico da rede conhecerá a identidade dos demais nós críticos existentes.

Algoritmo 8 COMPARECRITICALNODE()

```
1: while TRUE do
2:     waitFor associatedCriticalNodeIdentified OR Received compareCriticalNodeIdMSG OR Received
       newCriticalNodeIdMSG
3:     if associatedCriticalNodeIdentified then
4:         compareCriticalNodeIdMSG ← createNewCriticalNodeIdMSG(associatedCriticalNode)
5:         send compareCriticalNodeIdMSG to All direct neighboors
6:         for all Enqueued compareCriticalNodeIdMSG do
7:             if associatedCriticalNode ≠ compareCriticalNodeIdMSG.criticalNode then
8:                 newCriticalNodeIdMSG ← createNewCriticalNodeIdMSG(compareCriticalNodeIdMSG.criticalNode)
9:                 send newCriticalNodeIdMSG to associatedCriticalNode
10:            end if
11:        end for
12:    end if
13:    if Received compareCriticalNodeIdMSG then
14:        if associatedCriticalNode is Defined then
15:            if associatedCriticalNodeIdentified ≠ compareCriticalNodeIdMSG.criticalNode then
16:                newCriticalNodeIdMSG ← createNewCriticalNodeIdMSG(compareCriticalNodeIdMSG.criticalNode)
17:                send newCriticalNodeIdMSG to associatedCriticalNode
18:            end if
19:        else
20:            Enqueue compareCriticalNodeIdMSG
21:        end if
22:    end if
23:    if Received newCriticalNodeIdMSG AND thisNode.isCritical then
24:        if newCriticalNodeIdMSG.Id NOT in criticalNodeList then
25:            send criticalNodeList to newCriticalNodeIdMSG.Id
26:            send newCriticalNodeIdMSG to All nodes in criticalNodeList
27:            add newCriticalNodeIdMSG.Id to criticalNodeList
28:        end if
29:    end if
30: end while
```

4. Resultados Experimentais

Nesta seção analisamos o desempenho do DANCE em termos de efetividade para avaliação de centralidade e complexidade para aplicabilidade prática. Também ilustramos a aplicabilidade de nossa proposta DANCE em uma rede de larga escala.

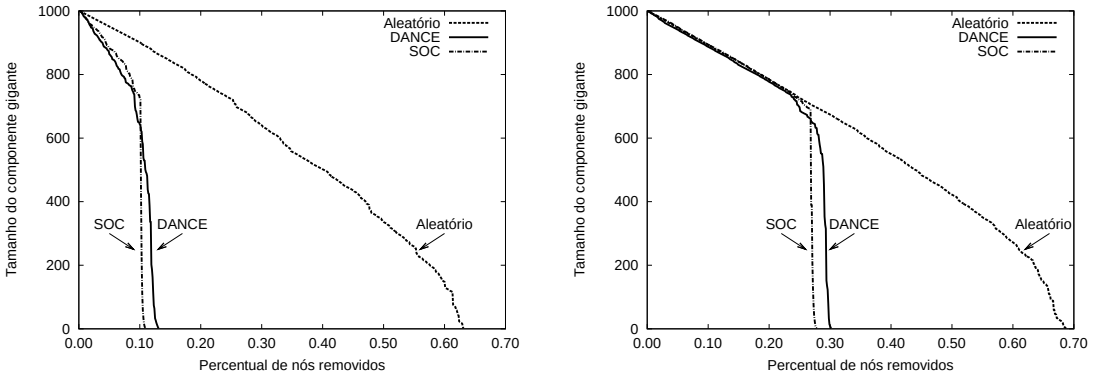
4.1. Efetividade

Nós comparamos a efetividade do DANCE com outro algoritmo recentemente proposto chamado SOC (*Second Order Centrality*) [Kermarrec et al., 2011] para avaliação distribuída de centralidade que pode ser usado em redes onde a topologia é desconhecida. SOC baseia-se em um caminho aleatório perpétuo onde se mede o desvio padrão do intervalo de tempo entre visitas consecutivas do caminho aleatório a cada nó. Apesar de eficaz para avaliação distribuída de centralidade, SOC apresenta dificuldades para determinar o número de passos necessários para que se obtenha valores consistentes de centralidade para todos os nós da rede. O número de passos deve ser grande o suficiente para que o caminho aleatório visite cada nó da rede um número de vezes suficientemente grande para que o desvio padrão do tempo entre as visitas possa ser bem determinado. Como a qualidade da determinação desse parâmetro depende do número de amostras, é de se esperar que sejam necessárias várias visitas do caminho aleatório a cada um dos nós da rede. Também é importante notar que o número de passos necessários para haver convergência depende da topologia da rede.

Na avaliação realizada foram utilizados dois tipos de redes sintéticas com 1000 nós: redes de escala-livre e aleatórias. As redes de escala-livre seguem o modelo Barabási-Albert (BA) [Barabási e Albert, 1999] considerando em sua geração duas conexões por cada novo nó conectado. Essas redes BA por construção sempre tem 1996 arestas, tendo portanto um grau médio igual à 3,992. As redes aleatórias seguem o modelo Erdös-Renyi (ER) [Erdos e Renyi, 1959] com probabilidade de conexão de 0,0045 (para grau médio ser similar ao das redes BA). Nessas redes ER o número de arestas não é fixo e depende de um processo aleatório de conexões entre os nós. Entretanto, devido ao número de nós e a probabilidade utilizada, o grau médio destas redes é aproximadamente 4.

Analisamos a efetividade da avaliação de centralidade dos nós da rede da seguinte maneira. Consideramos 100 redes sintéticas de 1000 nós seguindo o modelo BA e outras 100 redes seguindo o modelo ER, como descrito anteriormente. A partir de cada rede considerada, removemos o nó de maior centralidade indicado por cada algoritmo e avaliamos o impacto dessa remoção sobre a rede, verificando o número de nós restantes no maior componente conexo resultante — chamado de componente gigante. Espera-se que nós de alta centralidade — por definição, os mais importantes da rede ou nós críticos — indicados por cada algoritmo quando removidos causem a uma maior degradação da rede que se reflete na diminuição do componente gigante. O algoritmo DANCE prevê que se possa usar diferentes métricas para avaliação da centralidade na $2\_vizinhança$ de cada nó, possibilitando a obtenção de diferentes tipos de centralidade. Nessa avaliação utilizamos por simplicidade, porém sem perda de efetividade, a métrica de calcular o volume de cada $2\_vizinhança$ para a determinação da medida de centralidade em cada nó.

Na Figura 1 apresentamos o impacto da remoção de nós sobre o componente gigante de uma rede BA e uma rede ER representativas considerando a avaliação de centralidades oriundas das abordagens DANCE e SOC. Para a rede BA, utilizando o algoritmo DANCE para determinar os nós críticos a serem removidos, o componente gigante ficou reduzido a 20% de seu tamanho original — rede *fragmentada* — após 11,9% dos nós serem removidos. Utilizando o algoritmo SOC, o mesmo resultado foi obtido após 10,7% de nós removidos. No caso da rede ER, foram necessárias 28,9% dos nós re-

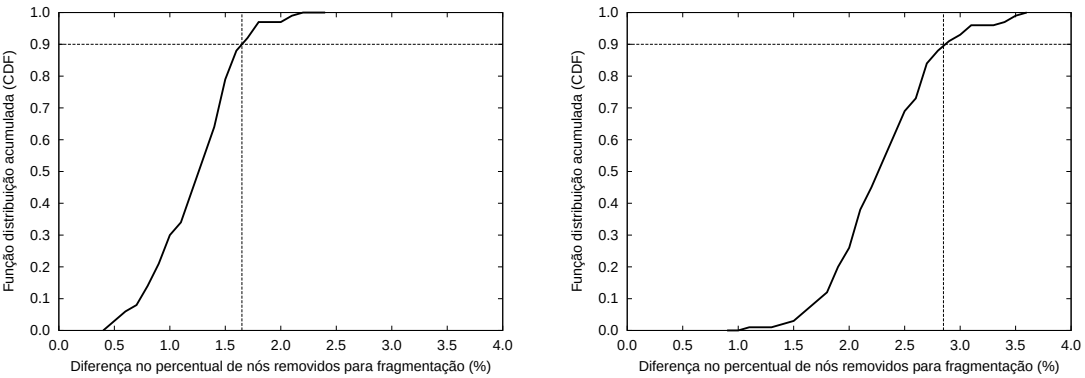


(a) Rede BA (b) Rede ER

Figura 1. Impacto da remoção de nós sobre o componente gigante da rede.

movidos utilizando o DANCE e 27,1% dos nós removidos utilizando o SOC. Para fim de comparação, incluímos ainda o impacto sobre o componente gigante obtido pela remoção aleatória dos nós da rede, demonstrando a efetividade do ataque realizado considerando as centralidades avaliadas tanto pelo SOC e DANCE. É importante notar que a efetividade do DANCE e SOC é bastante similar com DANCE necessitando de um pequeno número de nós removidos a mais para alcançar impacto igual do SOC no componente gigante.

A mesma análise do impacto da remoção de nós sobre o componente gigante foi realizada para 100 redes BA e 100 redes ER. Para todos os casos, os resultados são similares aos apresentados na Figura 1 para as duas redes BA e ER representativas. Na Figura 2 apresentamos a CDF da diferença entre o percentual necessário de nós removidos usando DANCE e SOC para fragmentação da rede (ou seja, redução do componente gigante a 20% do tamanho original). Observamos na Figura 2(a) que em 90% dos casos de redes BA a proposta DANCE necessita de menos de 1,6% de nós removidos a mais que SOC para fragmentar a rede; em 100% dos casos essa diferença é inferior a 2,5%. Para redes ER (Figura 2(b)), essa diferença é inferior a 2,9% e 3,6% em 90% e 100% dos casos, respectivamente. DANCE, portanto, alcança consistentemente uma efetividade bastante similar ao SOC ao avaliar a centralidade de nós da rede distribuidamente. No entanto, na Seção 4.2, mostraremos que essa efetividade similar é alcançada com uma complexidade de aplicabilidade significativamente menor do DANCE em relação ao SOC.



(a) Redes BA (b) Redes ER

Figura 2. Diferença entre DANCE e SOC no percentual de nós removidos para fragmentação.

4.2. Complexidade de aplicabilidade

Nesta subseção analisamos a complexidade para aplicação dos algoritmos SOC e DANCE na obtenção dos resultados de efetividade na avaliação de centralidade apresentados na Seção 4.1. Consideramos a complexidade de mensagens, tempo de convergência e complexidade computacional para obtenção dos resultados:

- **Complexidade de mensagens:** Como o algoritmo SOC somente calcula o valor de centralidade de cada nó da rede sem oferecer uma maneira para localizar nós de alta centralidade, em nossa avaliação consideraremos no DANCE apenas o número de mensagens utilizadas na fase de determinação da centralidade de cada nó. Nesse caso, a complexidade de mensagens esperada é de $O(n \times d_m^2)$ mensagens, onde n é o número de nós e d_m é o grau médio na rede. Consideramos as mesmas 100 redes BA e 100 redes ER da Seção 4.1. Kermarrec et al. [2011] indica que o uso de um único caminho aleatório torna difícil o estabelecimento de um critério de parada para o cálculo da centralidade e analisam que no pior caso $O(n^3)$ passos (mensagens), onde n é o número de nós na rede, são necessários para a avaliação distribuída de centralidade no SOC. Assim, para aplicação de SOC em redes de 1000 nós, Kermarrec et al. [2011] arbitram 2×10^6 passos (mensagens) do caminho aleatório. Logo, nós consideramos em nossas redes de 1000 nós a centralidade do SOC após 2×10^6 passos (mensagens) do caminho aleatório como [Kermarrec et al., 2011]. Em contraste, experimentalmente, o número médio de mensagens utilizadas pelo DANCE para calcular a centralidade de todos os 1000 nós de cada uma dessas redes foi de 47691 mensagens com desvio padrão de 3834 para redes BA e 29913 mensagens com desvio padrão de 1087 para redes ER. A variância apresentada decorre do fato que o número específico de mensagens utilizadas pelo DANCE depende da topologia da rede bem como também explica o maior número de mensagens e a maior variância em redes BA de escala livre. Na Figura 3 apresentamos esses resultados.

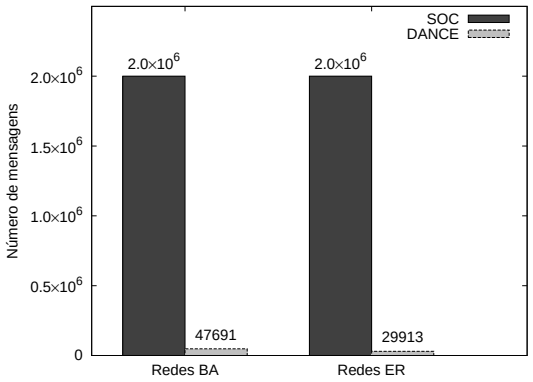


Figura 3. Número de mensagens para avaliação de centralidade.

- **Tempo de convergência:** No algoritmo DANCE o processo de troca de mensagens para determinação dos valores de centralidade é altamente paralelizado, sendo necessário apenas que as mensagens geradas por cada nó se propaguem por dois passos, fazendo com que o tempo necessário para conclusão do algoritmo seja limitado a dois passos ($O(1)$) e independente do número de nós da rede. Como o algoritmo SOC é baseado em um único caminho aleatório, todas as mensagens serão *sequenciais* no tempo, fazendo com que o tempo de execução do algoritmo

seja consideravelmente mais longo e dependente do número de nós. De acordo com Kermarrec et al. [2011], a complexidade de passos do SOC é $O(n^3)$, onde n é o número de nós da rede.

- **Complexidade computacional:** No caso do algoritmo DANCE utilizando a métrica adotada neste trabalho, o cálculo necessário se resume à determinação do volume da $2\_vizinhança$ de cada nó da rede. No caso do algoritmo SOC, é necessário calcular o desvio padrão do tempo entre as visitas sucessivas do caminho aleatório a cada nó da rede. Em ambos os casos, podemos notar que o custo computacional é modesto e totalmente distribuído entre os nós da rede.

4.3. Aplicabilidade do DANCE em redes de larga escala

De forma a ilustrar a aplicabilidade do algoritmo DANCE em redes de maior escala realizamos um experimento com um registro (*trace*) de rede real disponibilizado pela CAIDA[2] que representa uma amostra da conectividade ao nível de roteadores da Internet. Essa rede possui 190.914 nós, diâmetro 26, grau médio 6,34 e o nó de maior grau tem 1071 conexões; além de apresentar características de escala livre na distribuição do grau dos nós (ou seja, similar às redes sintéticas seguindo o modelo BA). É importante ressaltar que em uma rede de tal dimensão a aplicabilidade do algoritmo SOC [Kermarrec et al., 2011], apesar de distribuído, se torna inviável por sua alta complexidade de mensagens e elevado tempo de convergência, ambos de ordem $O(n^3)$. Em contraste, para calcular os valores de centralidade de todos os nós da rede utilizando o algoritmo DANCE foram necessárias aproximadamente 250 mensagens em média por nó. A Figura 4 apresenta a fragmentação do componente gigante dessa rede após remoção de nós indicados como de maior centralidade pelo DANCE e também uma remoção aleatória de nós para referência. Observamos comportamento similar ao obtido para redes BA sintéticas (cf. Figura 1(a)). Constatamos, portanto, que é possível aplicar o algoritmo DANCE a redes reais de maior porte obtendo os valores de centralidade efetivos para todos os nós utilizando um número relativamente pequeno de mensagens, em contraste com o estado da arte da literatura.

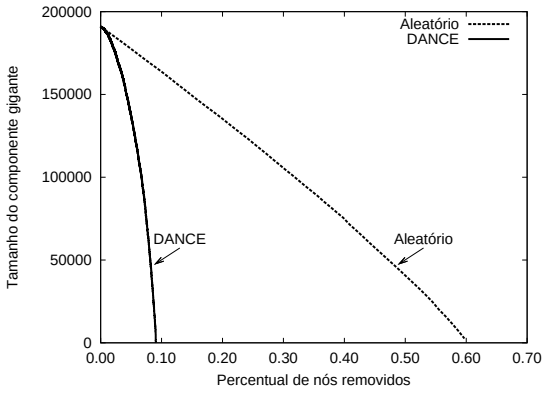


Figura 4. Algoritmo DANCE aplicado à fragmentação de uma rede real.

5. Conclusão

Neste trabalho apresentamos o algoritmo DANCE para avaliação e localização distribuída de centralidade de redes. As principais contribuições de nossa proposta são o cálculo descentralizado de centralidades de forma eficiente, podendo ser facilmente adaptado para

[2] http://www.caida.org/tools/measurement/skitter/router_topology/

obtenção de diferentes tipos de centralidade, e também um método para possibilitar a localização dos nós mais centrais da rede. Ambas as contribuições são efetuadas utilizando apenas informações da vizinhança a 2 saltos de cada nó sem a necessidade de conhecimento da topologia completa da rede. Isso é particularmente relevante considerando-se as dimensões das redes complexas em investigação atualmente. Nós avaliamos o algoritmo DANCE experimentalmente através de sua comparação com outro algoritmo recente de mesmo propósito. DANCE mostra-se simples, eficaz e eficiente, pois alcança uma efetividade na avaliação de centralidade de forma distribuída comparável ao estado da arte, porém com custo para sua aplicabilidade prática significativamente menor.

Dada a contribuição básica do DANCE em permitir avaliação e localização de centralidade de rede de forma distribuída, isso abre perspectivas para diferentes trabalhos futuros. Primeiro, podemos considerar o uso de diferentes definições de centralidade para calcular a centralidade local e avaliar seu impacto ao comparar com centralidades tradicionais. Segundo, dada a possibilidade de aplicar DANCE em redes de maior escala, pretendemos investigar redes reais distintas de larga escala, tais como redes sociais online e redes de distribuição de conteúdo, para analisar como a noção de centralidade pode trazer novos conhecimentos sobre a estrutura e comportamento dessas redes. Terceiro, a grande maioria das redes complexas atuais são dinâmicas. Pretendemos investigar como o algoritmo DANCE pode auxiliar na análise do comportamento dinâmico dessas redes através da avaliação e localização de centralidade.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPERJ, CNPq e MCT.

Referências

Barabási, A. e Albert, R. (1999). Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512.

Borgatti, S. (2005). Centrality and network flow. *Social Networks*, 27(1):55–71.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177.

Brin, S. e Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference (WWW 1998)*.

Dinh, T., Xuan, Y., Thai, M., Park, E., e Znati, T. (2010). On approximation of new optimization methods for assessing network vulnerability. In *Proc. of the IEEE INFOCOM*. IEEE.

Erdos, P. e Renyi, A. (1959). On random graphs I. *Publ. Math. Debrecen*.

Freeman, L. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41.

Kermarrec, A.-M., Le Merrer, E., Sericola, B., e Trédan, G. (2011). Second order centrality: Distributed assessment of nodes criticity in complex networks. *Computer Communications*, 34(5):619–628.

Lehmann, K. e Kaufmann, M. (2003). Decentralized algorithms for evaluating centrality in complex networks. Technical Report WSI-2003-10, Wilhelm Schickard Institut.

Nanda, S. e Kotz, D. (2008). Localized Bridging Centrality for Distributed Network Analysis. In *Proc. of Int. Conference on Computer Communications and Networks (ICCCN)*. IEEE.