Utilização da Árvore SPQR para um Cálculo Mais Eficiente das Medidas de Conectividade Baseadas em Cortes de Vértices

Henrique Hepp[1], **Jaime Cohen**[2], **Elias P. Duarte Jr.**[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19018 – CEP 81531-980 – Curitiba – PR – Brazil.

[2]Departamento de Informática – Universidade Estadual de Ponta Grossa (UEPG)
Av. General Carlos Cavalcanti 4748 – Bloco L - Sala 103 CEP 84030-900 – PR – Brazil.

hhepp@inf.ufrpr.br, jaimecohen@gmail.com, elias@inf.ufpr.br

Abstract. *The connectivity measure $\kappa_2(v)$ of a vertex v of graph G is the maximum number of vertex-disjoint paths between v and any other vertex of the graph. The purpose of this work is to improve the efficiency for computing $\kappa_2(v)$. A pre-processing step is proposed in which a SPQR tree is buit that identifies all tri-connected components of the graph. Thus, $\kappa_2(v)$ can be computed separately for each component. The proposal was implemented and tested in graphs that model concrete cases. Experimental results are presented showing that for graphs for which the proposed strategy does improve the efficiency for computing $\kappa_2(v)$, in comparison with the original approach.*

Resumo. *A medida de conectividade $\kappa_2(v)$ de um vértice v em um grafo G é o número máximo de caminhos vértices disjuntos que há entre esse vértice e qualquer outro vértice desse grafo. A proposta desse trabalho é melhorar a eficiência do cálculo de $\kappa_2(v)$ através de um pré-processamento do grafo. Nessa etapa, é construída uma árvore SPQR que identifica as componentes triconexas do grafo. Dessa forma, o $\kappa_2(v)$ pode ser calculado separadamente para cada componente. A proposta foi implementada e testada em grafos que modelam casos concretos. Apresentamos resultados para grafos para os quais observou-se que o tempo de cálculo de com o pré-processamento é significativamente menor que o tempo para calcular $\kappa_2(v)$ no grafo original.*

1. Introdução

Nesse trabalho investigamos uma propriedade dos grafos que é a conectividade. No contexto desse trabalho, a conectividade é quantificada por meio de medidas que indicam o quão conectado está um vértice com o restante do grafo [Cohen et al. 2011, Pires et al. 2011]. Essas medidas indicam o quão resistentes são esses vértices para serem desconectados do grafo com a retirada de arestas ou vértices.

As medidas de conectividade utilizadas no trabalho são a 2-aresta-conectividade, $\lambda_2(v)$, e a 2-vértice-conectividade, $\kappa_2(v)$ propostas em [Cohen 2013]. Essas medidas correspondem ao número máximo de caminhos arestas/vértices disjuntos que há entre um vértice v e qualquer outro vértice desse grafo. O caminho é uma sequência de vértices em que de cada vértice há uma aresta para o próximo da sequência.

Essas medidas de conectividade podem ser aplicadas para tolerância a falhas em redes. Um exemplo é o caso de uma rede de computadores na qual deseja-se posicionar

vários servidores de acordo com a melhor configuração possível para que a comunicação entre dois dos servidores dificilmente seja interrompida devido a falhas da rede. Para tal, deve-se achar uma configuração em que os $\lambda_2(v)$ e $\kappa_2(v)$ desses servidores sejam os maiores possíveis.

A medida 2-vértice-conectividade tem um custo computacional alto, pois para calcular a medida é necessário calcular os menores cortes de vértices entre todos os pares de vértices do grafo. Um corte de vértices é um conjunto de vértices cuja remoção desconecta o grafo. O custo alto pode tornar impraticável o uso da medida de conectividade baseada em corte de vértices.

Para diminuir o custo, esse trabalho propõe melhorar a eficiência do cálculo da medida 2-vértice-conectividade através de um pré-processamento do grafo. O objetivo é separar o grafo em componentes vértice-biconexas e vértice-triconexas para que $\kappa_2(v)$ possa ser calculado separadamente para cada componente, que tem número de vértices menor que o do grafo original. Para fazer essa separação é usada a árvore SPQR, descrita a seguir.

A árvore SPQR [Gutwenger 2010] visa decompor o grafo em suas componentes vértice-triconexas. Essa árvore contém quatro tipos de nós: nó tipo S, onde os vértices formam um ciclo; nó tipo P, formado por dois vértices com três ou mais arestas em paralelo; nó tipo Q, o caso trivial com apenas uma aresta; e nó tipo R, uma componente triconexa que não corresponde aos outros casos.

Nesse trabalho, foi implementado um sistema que primeiro calcula as componentes biconexas de um grafo recebido como entrada. Em seguida, é executado um programa que recebe como entrada essas componentes e as transforma em árvores SPQR. No próximo passo, cada componente das árvores SPQR geradas é repassada para outro programa que calcula o $\kappa_2(v)$. Por último, como um vértice pode estar em mais de uma componente, os vértices que receberam mais de um $\kappa_2(v)$ ficam com o maior valor da medida de conectividade.

A proposta foi implementada e testada em grafos reais, como UsaAir97 [Batagelj and Mrvar 2006], o grafo que representa a rede de aeroportos dos EUA. Nos resultados obtidos observou-se que nos grafos utilizados existe uma componente biconexa principal, com a maior parte dos vértices, e, a partir dessa componente, pode ser obtida uma componente triconexa. Nesses casos, o tempo computacional para o cálculo de $\kappa_2(v)$ corresponde principalmente ao cálculo dessa componente triconexa, o que é significativamente menor, no caso de UsaAir97 foi de 68%, que o tempo para calcular $\kappa_2(v)$ no grafo original.

O restante deste trabalho está organizado da seguinte maneira. A seção 2 descreve as medidas de conectividade a seção 3 descreve informalmente a árvore SPQR e a sua aplicação no cálculo da medida $\kappa_2(v)$. A seção 4 descreve a implementação e os resultados experimentais. Por fim, a seção 5 conclui o trabalho.

2. Medidas de Conectividade

Existem diversas medidas para caracterizar o quão conectado está um vértice de um grafo [Nagamochi and Ibaraki 2008, Cohen 2013]. Uma das medidas de conectividade de vértices é o próprio grau. Em um grafo $G = (V, E)$, onde V é o conjunto de vértices e

E o conjunto de arestas, o *grau* de um vértice v é igual ao número de arestas incidentes a v e é denotado por grau(v). Quando o grau de um vértice tem um valor alto, geralmente ele está bem conectado com o grafo. Entretanto nem sempre é assim, por exemplo, considere o caso de um grafo que contém um subgrafo com topologia estrela. Veja a figura 1. O vértice 1, embora tenha um grau alto, será desconectado do grafo com a remoção do vértice 2. Nesse caso, um vértice tem um grau alto, mas é desconectado do grafo pela remoção de apenas um vértice ou aresta. Dessa forma, um grau alto não garante que um vértice esteja bem conectado.

Para termos mais informações sobre a conectividade de um vértice podemos usar outras medidas, como a medida de conectividade baseada em corte de arestas e a baseada em corte de vértices, descritas nas próximas seções.

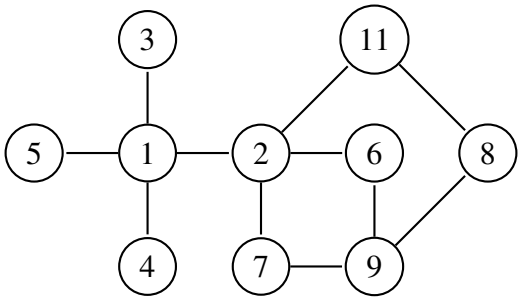

Figura 1. Grafo com o vértice 1 com topologia estrela, ele é desconectado do grafo retirando o vértice 2.

2.1. Medidas de Conectividade Baseadas em Cortes de Arestas

A medida de conectividade de vértices baseada em cortes de arestas [Duarte et al. 2004, Cohen et al. 2011] também chamada de i-aresta-conectividade é definida da seguinte forma:

Definição 1 *Seja $G = (V, E)$ um grafo não direcionado. Considere um conjunto de vértices $X \subseteq V, |X| \geq 2$. A aresta-conectividade de X em relação a G é o tamanho de um corte mínimo que separa quaisquer pares de vértices em X. Denotamos a aresta-conectividade de X por $\lambda(X)$. Se $|X| = 1$, com $X = \{v\}$, então definimos $\lambda(X) = grau(v)$.*

De acordo com essa definição, em grafos com arestas com capacidades unitárias, $\lambda(X)$ é um número que indica o menor número de arestas a serem retiradas de um grafo G para desconectar algum par de vértices de um conjunto X.

Definição 2 *A i-aresta-conectividade de um vértice v, denotada por $\lambda_i(v)$, é a máxima aresta-conectividade de um conjunto $X \subseteq V$ que satisfaz:*

i. $v \in X$, *e*
ii. $|X| \geq i$

Em particular, em grafos com capacidades unitárias, a medida $\lambda_2(v)$ de um vértice v é o maior número de caminhos aresta disjuntos entre v e qualquer outro vértice. A figura 2 mostra um exemplo. O vértice 1 tem 5 caminhos aresta disjuntos com o vértice 5, como o vértice 1 não tem um número maior de caminhos aresta disjuntos com qualquer outro vértice, $\lambda_2(1) = 5$.

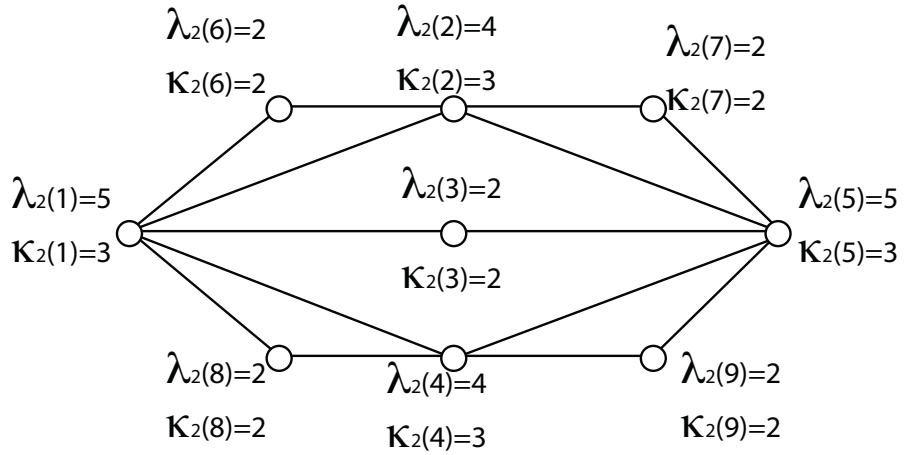

Figura 2. Grafo com os valores da 2-aresta-conectividade e da 2-vértice-conectividade [Pires et al. 2011, Pires 2011].

2.2. Medidas de Conectividade Baseadas em Cortes de Vértices

A medida de conectividade de vértices baseadas em corte de vértices, vértice-conectividade [Pires 2011, Pires et al. 2011] é definida a seguir:

Definição 3 *Dados dois vértices $s, t \in V$, a vértice-conectividade entre s e t, denotada por $\kappa(s,t)$, é o número de caminhos vértice disjuntos entre s e t*

Definição 4 *A vértice-conectividade de um conjunto $X \subseteq V$ é a menor vértice-conectividade entre quaisquer pares de vértices em X e é denotada por $\kappa(X)$.*

Definição 5 *Definimos a i-vértice-conectividade de um vértice v, denotada por $\kappa_i(v)$, como a maior vértice-conectividade de um conjunto $X \subseteq V$ satisfazendo:*

i. $v \in X$, *e*
ii. $|X| \geq i$

Em particular, a medida $\kappa_2(v)$ de um grafo G é o maior número de caminhos vértices disjuntos entre v e qualquer outro vértice de G. A figura 2 mostra um exemplo. O vértice 1 tem 3 caminhos vértice disjuntos com o vértice 5, como o vértice 1 não tem um número maior de caminhos vértice disjuntos com qualquer outro vértice, $\kappa_2(1) = 3$.

2.3. Algoritmo para Calcular a Medida de Conectividade Baseada em Cortes de Vértices

Para calcular a medida $\kappa_2(v)$ de cada vértice de um grafo $G = (V, E)$, o algoritmo proposto em [Pires et al. 2011, Pires 2011] requer o cálculo de $|V| - 1$ cortes mínimos entre pares de vértices. Para calcular esses cortes de vértices reduz-se o problema de corte de vértices ao corte de arestas. Essa redução é ilustrada nas figuras 3, 4 e 5.

No primeiro passo, mostrado na figura 3, o grafo não orientado é transformado em um grafo orientado. Para isso, as arestas do grafo original são duplicadas de modo que uma aresta $\{a, b\}$ corresponda a duas arestas direcionadas $\{(a, b), (b, a)\}$.

No segundo passo, na figura 4 é mostrado um subgrafo do grafo da figura 3, cada vértice v é duplicado para os vértices v' e v''. As arestas que tinham como origem o vértice

v, têm agora como origem o vértice v''. As arestas que tinham como destino o vértice v têm agora como destino o vértice v'. Por último, adiciona-se as arestas direcionadas (v', v'') entre todos os pares de vértices associados.

No terceiro passo, mostrado na figura 5, para garantir que o corte mínimo utilize arestas que correspondem ao grafo original atribui-se as seguintes capacidades às arestas: as arestas do tipo (v', v'') recebem capacidade de valor 1 e as demais arestas recebem capacidade $|V| - 1$.

Nesse grafo resultante é utilizado o algoritmo para calcular o fluxo máximo, que retorna o corte mínimo de arestas, que corresponde ao corte mínimo de vértices do grafo original.

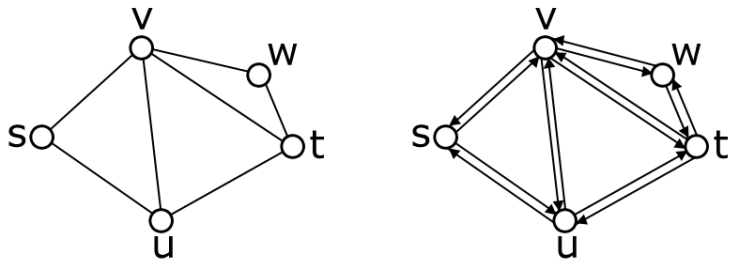

Figura 3. Transformação de grafo não orientado para orientado [Pires 2011].

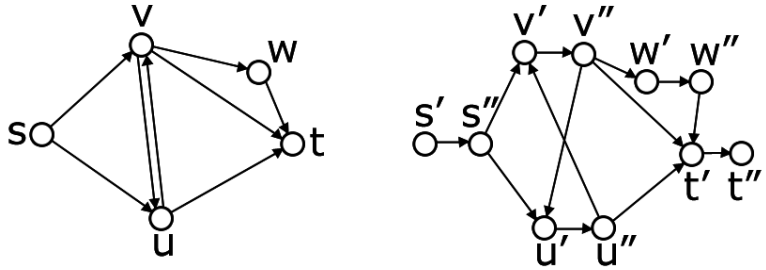

Figura 4. Transformação de grafo orientado para aplicação de corte de arestas [Nagamochi and Ibaraki 2008].

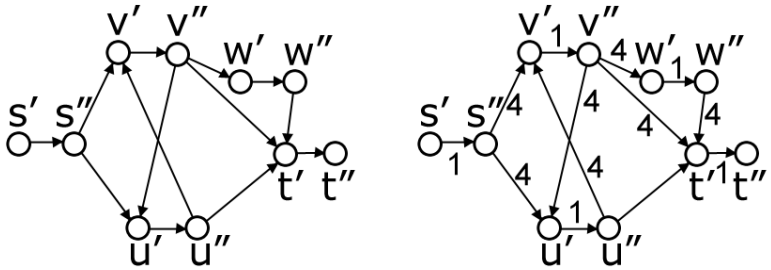

Figura 5. Aplicação de pesos para corte de arestas [Nagamochi and Ibaraki 2008].

2.3.1. Melhoria da Eficiência do Cálculo de $\kappa_i(v)$

Para reduzir o tempo de cálculo de $\kappa_i(v)$ podemos usar o seguinte lema [Pires et al. 2011, Pires 2011]:

Lema 1 *Dado um grafo $G = (V, E)$ e $v \in V$, tem-se que $\kappa_i(v) \leq \lambda_i(v)$ para todo $i \geq 2$.*

Esse lema parte da relação de que o número de caminhos vértice disjuntos entre dois vértices v e w é limitado pelo número de caminhos aresta disjuntos entre esses dois vértices.

Como o cálculo de $\lambda_i(v)$ é computacionalmente mais rápido, podemos usar esse fato no cálculo de $\kappa_i(v)$. De acordo com o lema 1, o $\lambda_i(v)$ é o limitante superior de $\kappa_i(v)$. Portanto, ao calcular os fluxos máximos entre v e os outros $|V| - 1$ vértices, o cálculo poderá ser interrompido no momento em que o número de caminhos vértice disjuntos alcançar o valor de $\lambda_i(v)$.

3. Utilização da Árvore SPQR no Cálculo de $\kappa_2(v)$

A árvore SPQR é uma árvore que contém as componentes vértice triconexas de um grafo vértice biconexo. Devido ao fato da definição e do algoritmo com tempo linear de construção dessa árvore, definidos formalmente em [Hopcroft and Tarjan 1972, Battista and Tamassia 1996, Gutwenger 2010], serem bastante complicados, não os apresentamos nesse artigo. Expomos apenas uma definição informal com as informações necessárias para a compreensão desse trabalho.

As árvores SPQR são construídas para grafos biconexos. Grafos que não sejam biconexos serão inicialmente decompostos em suas componentes biconexas. Dessa forma, assumiremos no restante desta seção que o grafo é biconexo.

A figura 6 apresenta um exemplo de uma árvore SPQR. A árvore SPQR é composta de quatro tipos de nós, os nós S, P, Q e R, definidos abaixo:

1. Nó tipo S. Corresponde ao caso dito serial. É um ciclo com três ou mais vértices. Um ciclo de três vértice é uma componente triconexa. Quando o ciclo tem mais de três vértices é uma componente biconexa.
2. Nó tipo P. Corresponde ao caso dito paralelo. É um grafo com dois vértices e 3 ou mais arestas entre eles.
3. Nó tipo Q. Corresponde ao caso dito trivial com apenas uma aresta. Várias implementações, como a usada nesse trabalho, omitem esse nó.
4. Nó tipo R. Corresponde ao caso dito rígido e é uma componente triconexa que não é do tipo S nem do tipo P.

Nas implementações mais recentes, as ligações entre os nós (S, P e R) são arestas virtuais, na figura 6 são as arestas pontilhadas. Essas arestas correspondem aos dois vértices que os dois nós compartilham.

Para esse trabalho, duas observações que decorrem da definição formal ainda são importantes:

1. Não existem dois nós vizinhos do mesmo tipo, pois nesses casos eles formariam apenas um nó.
2. Como pode-se ver pela figura 6, o nó P ocorre entre os nós S e R por existir uma aresta no grafo original entre os dois vértices do corte de vértices. Por outro lado, se houverem dois nós R e S vizinhos, então os dois vértices do corte de vértices não podem estar conectados por uma aresta no grafo original.

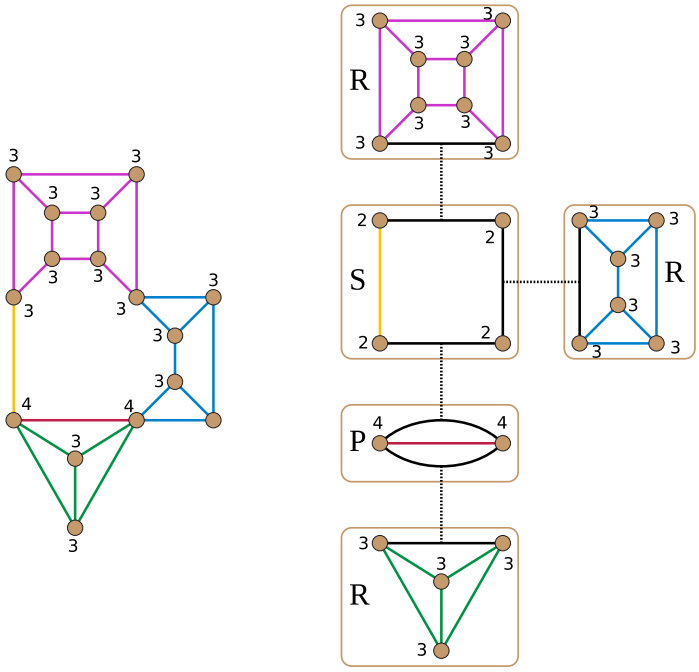

Figura 6. Exemplo de uma árvore SPQR com os valores de $\kappa_2(v)$.

3.1. Utilização da Árvore SPQR no Cálculo de $\kappa_2(v)$

Para melhorar a eficiência do cálculo $\kappa_2(v)$ para todo vértice de um grafo pode-se usar a árvore SPQR para identificar as componentes triconexas e depois calcular o $\kappa_2(v)$ separadamente para cada componente S, P, e R. Cada componente é separada do restante do grafo por cortes de dois vértices, enquanto que dentro da componente, os vértices podem ter entre eles múltiplos caminhos vértice disjuntos. Assim basta calcular o $\kappa_2(v)$ individualmente para cada componente.

Deve-se lembrar que há vértices que estão em mais de uma componente. O $\kappa_2(v)$ desses vértices pode ser calculado comparando-se entre eles o $\kappa_2(v)$ obtido em cada componente. O maior $\kappa_2(v)$ é o $\kappa_2(v)$ do grafo inteiro.

Observando as componentes de forma individual, podemos dizer que:

Componente S É o caso em que os vértices formam um ciclo, o $\kappa_2(v)$ para cada um desses vértices é, portanto, 2. Pois só há dois caminhos vértices disjuntos entre eles.

Componente R É uma componente triconexa que é repassada diretamente para o programa que faz o cálculo de $\kappa_2(v)$.

Componente P O $\kappa_2(v)$ dos vértices da componente P é o número de caminhos vértice disjuntos entre seus dois vértices no grafo original. Se formos comparar o grafo original com a árvore, veremos, por exemplo, pela figura 6, que entre os dois vértices de P existem 4 caminhos vértice disjuntos no grafo original. Já pela árvore, dentro da componente P existem apenas 3 caminhos vértice disjuntos. Isso mostra que para o cálculo de $\kappa_2(v)$ precisamos calcular o número de caminhos vértice disjuntos dos dois vértices de P com relação a todo grafo original, e não apenas dentro da componente P.

Depois de calcular o $\kappa_2(v)$ para cada componente, compara-se o $\kappa_2(v)$ dos vértices que estão em mais de uma componente. O maior $\kappa_2(v)$ entre eles é o $\kappa_2(v)$ do grafo. Veja por exemplo a figura 6, considere o vértice do canto inferior direito da componente S, $\kappa_2(v)$ = 2. Esse vértice também se encontra na componente R, canto inferior da esquerda com $\kappa_2(v)$ = 3, e na componente P, vértice da direita com $\kappa_2(v)$ = 4. O maior valor é o da componente P, por isso, o valor de $\kappa_2(v)$ desse vértice fica igual a 4.

4. A Implementação

Nesse trabalho, foi implementado um *script* em Python que primeiro calcula as componentes biconexas de um grafo recebido como entrada. Em seguida, o *script* chama um programa em Java que recebe como entrada essas componentes e as transforma em árvores SPQR. No próximo passo, cada componente das árvores SPQR gerada é repassada para outro programa, em C, que calcula o $\kappa_2(v)$. Por último, como um vértice pode estar em mais de uma componente, os vértices que receberam mais de um $\kappa_2(v)$ ficam com o maior valor da medida de conectividade. Dessa forma, a saída do *script* são os valores de $\kappa_2(v)$ do grafo recebido como entrada.

Os principais passos executados pelo *script* são os seguintes:

1. Recebe um grafo G como entrada
2. Calcula as componentes biconexas de G utilizando uma função da biblioteca NetworkX do Python
3. Para todo vértice v de componentes com apenas dois vértices, faz $\kappa_2(v) := 1$
4. Para cada componente biconexa, calcula a árvore SPQR usando a última versão (0.2.429) da biblioteca Java para grafos jBPT[1]
5. Para todo vértice v em componente S, faz $\kappa_2(v) := 2$
6. Para cada componente R, execute o programa em C escrito por Karine Pires[**?**, Pires 2011] que calcula $\kappa_2(v)$ para cada vértice v da componente
7. Para cada componente P, calcula o número de caminhos vértice disjuntos entre seus dois vértices usando o mesmo programa do passo anterior
8. Finalmente, compara o $\kappa_2(v)$ de cada vértice v existente em mais de uma componente, e faz $\kappa_2(v)_{final} :=$ o maior $\kappa_2(v)$

4.1. Resultados Experimentais

O pré-processamento de dados foi aplicado sobre grafos que modelam casos concretos: UsaAir97 [Batagelj and Mrvar 2006] representa os aeroportos dos EUA, Yeast [Bu et al. 2003] é uma rede de interações de proteínas, Rome [Storchi et al. 1999] representa as ruas de Roma, Geocomp2 [Batagelj and Mrvar 2006] e CA-GrQc [Kleinberg et al. 2007] são redes de colaboração científica e Powergrid [Watts and Strogatz 1998] é uma rede de distribuição de energia elétrica. Nas seções seguintes é apresentada uma síntese dos resultados.

4.1.1. Grafo: UsaAir97

Nessa seção são apresentados resultados experimentais obtidos para o grafo UsaAir97 que representa os aeroportos dos EUA. Foram feitas medidas para o cálculo de $\kappa_2(v)$

[1]Disponível em `https://code.google.com/p/jbpt/`

utilizando tanto o grafo original como dois tipos de pré-processamento: no primeiro, o cálculo é feito sobre as componentes biconexas, calculadas no passo dois do *script* acima, no segundo, o cálculo é feito utilizando as árvores SPQR.

Fazendo o pré-processamento do grafo UsaAir97, que tem 332 vértices e 2126 arestas, podemos observar que há, após passar a transformação em componentes biconexas, uma componente B com a maior parte dos vértices. Essa componente B tem o seu tamanho reduzido com a transformação da árvore SPQR para uma componente R. As demais componentes do grafo, sejam biconexas ou das árvores SPQR, têm um tamanho pequeno, de menos de 10 vértices. Essas componentes têm o $\kappa_2(v)$ calculado rapidamente.

Podemos visualizar esse fato na figura 7 e na tabela 1. De um grafo de 332 vértices, temos uma componente grande biconexa com 244 vértices, que são os vértices vermelhos junto com os vértices verdes. Os vértices em azul correspondem às demais componentes biconexas que para os quais o tempo de cálculo de $\kappa_2(v)$ é desprezível. Os vértices brancos formam a maior componente R, os vértices em verde correspondem às demais componentes da árvore SPQR que também têm tempos do cálculo dos $\kappa_2(v)$ desprezíveis. Fazendo o cálculo de $\kappa_2(v)$, observa-se que todos esses vértices para os quais os tempos de cálculo $\kappa_2(v)$ são desprezíveis têm valor de $\kappa_2(v)$ baixo (veja a figura 8). São os vértices com baixa conectividade no grafo. Desse modo, para o grafo da figura 7 observou-se que o pré-processamento foi efetivo. De acordo com a tabela 1, temos uma componente B com 73% do tamanho do grafo de entrada e uma componente R, da árvore SPQR, com 62% do tamanho do grafo de entrada.

O cálculo de $\kappa_2(v)$ foi feito em um computador com 8 GB de memória e com um processador Intel Core i7-3537U com 2.00 x 4. De acordo com a tabela 2, houve uma melhoria significativa no tempo de execução. Utilizando apenas componentes biconexas a melhoria é de 52% do tempo de execução, com a árvore SPQR a melhoria é de 68%.

Tabela 1. Tabela com o número de vértices de UsaAir97 após o pré-processamento. Ao lado está a porcentagem de vértices com relação aos vértices do grafo original.

Número de vértices	Grafo original	Maior componente B	Maior componente R
UsaAir97	332	244 73%	205 62%

Tabela 2. Tempo médio para 3 processamentos do cálculo de $\kappa_2(v)$ para todos os vértices do grafo UsaAir97.

Tempo grafo original	245s
Tempo componentes biconexas	117s 48% do original
Tempo árvore SPQR	79s 32% do original

4.1.2. Os Outros Grafos

Nessa seção são apresentados resultados experimentais obtidos para os grafos Yeast, Rome, Geocomp2, CA-GrQc e Powergrid. Como o número de vértices desses grafos

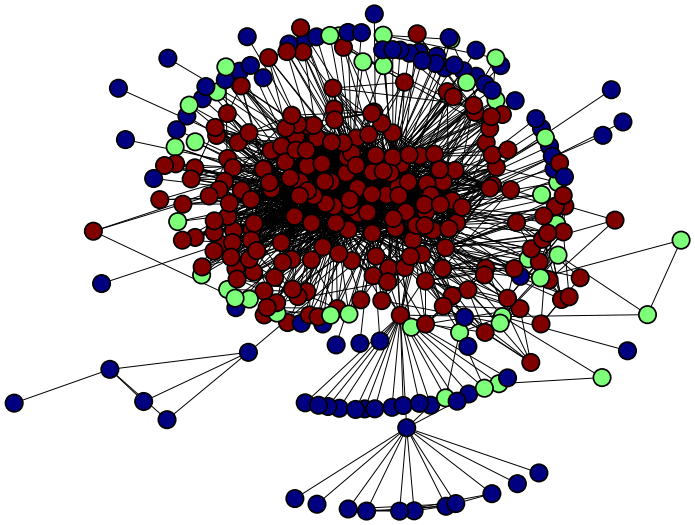

Figura 7. Grafo de UsaAir97, os vértices internos, em vermelho, correspondem ao maior componente R, a maior componente B corresponde aos vértices em vermelho junto com os verdes.

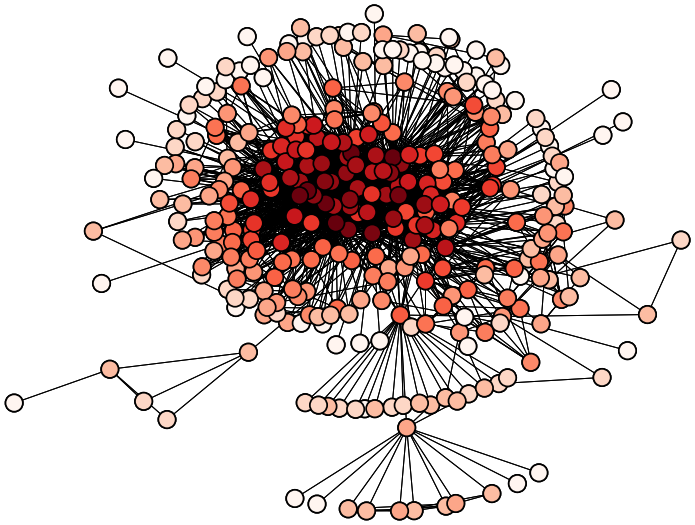

Figura 8. Grafo de UsaAir97, a tonalidade indica o valor de $\kappa_2(v)$, quanto mais escuro maior é o valor de $\kappa_2(v)$, os vértices em branco têm $\kappa_2(v) = 1$.

é maior que 1000, o tempo de processamento é muito grande (acima de 8 horas) para calcular o $\kappa_2(v)$. Por isso, foi apenas feita a construção da árvore SPQR e medido o tamanho das componentes obtidas.

Observou-se nesses grafos uma situação semelhante a observada no grafo UsaAir97, todos têm uma componente biconexa grande que é reduzida para uma componente R grande. Todas as demais componentes são pequenas e, portanto, têm tempos de cálculo de $\kappa_2(v)$ desprezíveis.

Pela tabela 3 pode-se verificar, de forma qualitativa, o potencial da redução de tempo de execução originada pelo pré-processamento. Com a árvore SPQR temos a menor redução de vértices para o grafo Rome, de apenas 33% com relação ao original. A maior redução foi obtida com o grafo Powergrid que foi de 79%.

Tabela 3. Tabela com o número de vértices após o pré-processamento. Ao lado está a porcentagem de vértices com relação aos vértices do grafo original.

Número de vértices	grafo original	maior componente B		maior componente R	
Yeast	2221	1464	66%	1137	51%
Rome	3353	2689	80%	2251	67%
Geocomp2	3621	1901	52%	1149	32%
CA-GrQc	4158	2651	64%	2456	59%
Powergrid	4941	3040	62%	1022	21%

5. Conclusão

Esse trabalho apresentou uma proposta para melhorar a eficiência do cálculo de $\kappa_2(v)$ através de um pré-processamento do grafo. Nessa etapa, o grafo é primeiro dividido em componentes biconexas e depois, para cada componente biconexa, é construída uma árvore SPQR que identifica as componentes triconexas do grafo. Dessa forma, o $\kappa_2(v)$ pode ser calculado separadamente para cada componente, que tem número de vértices menor que o do grafo original.

A proposta foi implementada e testada em grafos que modelam casos concretos. Nos resultados obtidos observou-se que nos grafos utilizados existe uma componente biconexa principal, com a maior parte dos vértices, e, a partir dessa componente, pode ser obtida uma componente triconexa tipo R. Para o grafo UsaAir97, com 332 vértices, que representa os aeroportos dos EUA obteve-se uma componente biconexa principal com 244 vértices e uma componente R com 205 vértices. Para o grafo Powergrid, com 4941 vértices, que representa uma rede de distribuição elétrica obteve-se uma componente biconexa com 3040 vértices e uma componente R com 1021 vértices. Nesses casos, o tempo computacional para o cálculo de $\kappa_2(v)$ corresponde principalmente ao cálculo de $\kappa_2(v)$ nessa componente triconexa tipo R, o que é significativamente menor que o tempo para calcular $\kappa_2(v)$ no grafo original. Para o grafo UsaAir97 obteve-se uma redução de 62% no tempo de cálculo de $\kappa_2(v)$ usando a árvore SPQR com relação ao cálculo de $\kappa_2(v)$ no grafo original.

Nesse trabalho, observou-se que a implementação da árvore SPQR pela biblioteca jBPT não é linear. Por isso, sugere-se para testes futuros uma biblioteca disponível em

C++ no site `www.ogdf.net/doc-ogdf/`. Um trabalho futuro a ser realizado é estudar o comportamento do algoritmo de cálculo do $\kappa_2(v)$ com a árvore SPQR diante da inserção ou remoção de vértices e arestas.

Referências

Cohen, J., Duarte, E. P. and Schroeder, J. (2011). Connectivity Criteria for Ranking Network Nodes, *Communications in Computer and Information Science (CCIS)*, Vol. 116. pp. 35-45.

Cohen, J. (2013). Algoritmos Paralelos para Árvores de Cortes e Medidas de Centralidade em Grafos, Tese de Doutorado, Departamento de Informática - Universidade Federal do Paraná,

Pires, K., Cohen, J. and Duarte, E. P. (2011). Medidas de Conectividade Baseadas em Cortes de Vertices para Redes Complexas, *12 Workshop de Testes e Tolerância a Falhas (WTF'2011), Anais do SBRC'2011*.

Pires, K. (2011). Medidas de Conectividade Baseadas em Cortes de Vértices para Redes Complexas, Dissertação de Mestrado, Departamento de Informática - Universidade Federal do Paraná.

Gutwenger, C. (2010). Application of SPQR-Trees in the Planarization Approach for Drawing Graphs, Tese de Doutorado, Technischen Universität Dortmund an der Fakultät für Informatik.

Batagelj, V. and Mrvar, A. (2006). Pajek datasets. Disponível em `http://vlado.fmf.uni-lj.si/pub/networks/data/`, Acessado em dezembro de 2015.

Nagamochi, H. and Ibaraki, T. (2008). *Algorithmic Aspects of Graph Connectivity*, Cambridge University Press.

Duarte, E. P., Santini, R. and Cohen, J. (2004). Delivering Packets During the Routing Convergence Latency Interval Through Highly Connected Detours, *DSN*, pp. 495–. IEEE Computer Society.

Hopcroft, J. E. and Tarjan, R. E. (1972). Finding the Triconected Components of a Graph, *Technical Report 72-140*, Cornell University.

Di Battista, G. and Tamassia, R. (1996). On-Line Maintenance of Triconnected Components with SPQR-Trees, *Algorithmica*, Vol. 15, pp. 302-318.

Bu, D., Zhao, Y., Cai, L., Xue, H., Zhu, X., Lu, H., Zhang, J., Sun, S., Ling, L., Zhang, N., Li, G., Chen, R. (2003). Topological Structure Analysis of the Protein-protein Interaction Network in Budding Yeast, *Nucleic acids research*, Vol. 31(9), pp. 2443–2450.

Storchi, G., Dell'Olmo, P. and Gentili, M. (1999) Directed Road Network of the City of Rome (1999). Disponível em `http://www.dis.uniroma1.it/challenge9/download.shtml`, acessado em dezembro de 2015.

Kleinberg, J. M., Leskovec J. and Faloutsos, C. (2007). Graph Evolution: Densification and Shrinking Diameters, *ACM Transactions on Knowledge Discovery from Data (ACM TKDD)*.

Watts, D. J. and Strogatz, S. H. (1998). Collective Dynamics of 'Small-World' Networks, *Nature*, Vol. 393(6684), pp. 440–442.