

# Cálculo de Sub-Redes e Pontos de Articulação

Relatório do 1º Projecto - Análise e Síntese de Algoritmos

Baltasar Dinis, 89416 e Afonso Ribeiro, 86752

**Resumo**—Uma rede deve ser desenhada de forma a que seja resiliente a falhas de componentes individuais, permitindo ao seu utilizador uma utilização contínua do sistema. A existência de pontos únicos de falha<sup>1</sup> compromete esta resiliência, impedindo que a rede consiga escalar, face ao tráfego. Neste relatório, expomos uma metodologia para automaticamente verificar se uma rede é resiliente e quantas sub-redes estão presentes na mesma. Este trabalho foi realizado no contexto da Unidade Curricular de Análise e Síntese de Algoritmos, no ano lectivo de 2018-2019.

## I. INTRODUÇÃO

Redes de computadores são hoje em dia ubíquas e essenciais para o funcionamento normal de empresas, do Estado e da sociedade em geral. Devido à sua importância, uma rede deve ser capaz de tolerar falhas individuais e independentes dos seus componentes, de forma a que um serviço que a utilize consiga escalar com o tráfego. Se a rede tiver um ponto que, ao falhar, desconecte a rede, não se pode considerar que a rede tem uma topologia resiliente.

É também útil conseguir saber quais são as sub-redes existentes numa dada topologia de roteadores, e quantas existiriam caso fossem retirados os pontos únicos de falha.

Neste relatório, apresentaremos a nossa solução para este problema. A estrutura do relatório é a seguinte. Em II, fazemos um mapeamento do problema para um problema de grafos e expomos a nossa solução. Em III, fazemos uma análise teórica do algoritmo empregue, nomeadamente em termos de complexidade. Em

IV apresentamos os resultados da nossa avaliação experimental e em V comentamos a correspondência entre os resultados experimentais e os valores teóricos.

## II. DESCRIÇÃO DA SOLUÇÃO

Representamos uma rede de roteadores como um grafo não dirigido. Notemos que uma sub-rede é uma componente do grafo, pois existem caminhos entre dois quaisquer roteadores de uma sub-rede, e não existe nenhum roteador atingível a partir de um outro na sub-rede que não pertença ele mesmo à sub-rede.

Mapeamos conceptualmente os pontos únicos de falha na noção de ponto de articulação. Na literatura [1], define-se um ponto de articulação como aquele cuja remoção causaria uma desconexão de um grafo conexo (consideramos as componentes como sub-grafos conexos). Note-se a seguinte, e interessante, propriedade dos pontos de articulação:

**Lema.** Seja  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  um grafo e  $a \in \mathcal{V}$ . Se existir um par de vértices  $u, v \in \mathcal{V}$  tal que existe um caminho de  $u$  para  $v$  que passa por  $a$ . Então  $a$  é um ponto de articulação se e só se todos os caminhos de  $u$  para  $v$  passarem por  $a$ .

**Dem.** Se existir um caminho de  $u$  para  $v$  que não passe por  $a$  então a remoção de  $a$  não faz com que a componente se desconecte. Se  $a$  é um ponto de articulação então a sua remoção terá de desconectar a componente. Isto só acontece se todos os caminhos que vão de  $u$  para  $v$  passarem por  $a$  (caso contrário a componente manter-se-ia conexa).

Note-se que esta propriedade é importante, em [2] é dada como a definição de

<sup>1</sup>Single Points of Failure em inglês

ponto de articulação, definindo a essência do ponto de articulação como ponto único de falha.

A nossa abordagem baseia-se na procura em profundidade<sup>2</sup>, empregando algoritmos definidos por Tarjan [2], [3]. Em cada uma das subsecções seguintes apresentamos como a DFS é aplicada para produzir cada um dos resultados pedidos. Importamos de [1] o seguinte lema:

**Lema.** (Complexidade Temporal da DFS) A pesquisa em profundidade é calculada em  $\Theta(V + E)$ , onde  $V$  e  $E$  denotam o número de vértices e arestas, respectivamente.

#### A. Cálculo de Sub-Redes

Uma sub-rede é uma componente do grafo. Como descrito em [3], podemos obter as componentes aplicando a DFS. Quando não existirem mais vértices a explorar, encontramos necessariamente uma componente. No contexto do problema, não temos que identificar a componente inteira, mas sim o identificador do roteador com maior índice.

Fazemos isso de forma eficiente impondo uma ordenação na extracção. Extraímos o roteador não visitado de maior índice, adicionando-o ao fim de uma lista que guarda os identificadores das sub-redes, pois é garantido que todos os roteadores de índice maior que o dele já tenham sido explorados - não sendo possível que façam parte desta componente. Como temos de imprimir a lista de identificadores de sub-rede por ordem crescente, percorremos a lista em ordem inversa.

#### B. Cálculo de Pontos de Articulação

Introduzimos os seguintes lemas .

**Lema.** Seja  $v \in \mathcal{V}$  tiver menos de duas adjacências não é um ponto de articulação. A demonstração é trivial.

**Lema.** Seja  $root \in \mathcal{V}$  a raiz de uma árvore gerada pela DFS.  $root$  é um

ponto de articulação se e só se tem pelo menos duas crianças na árvore. Não pode ser ponto de articulação se não tiver pelo menos duas crianças. Se tiver  $n$  crianças  $c_1, \dots, c_n$ . Se as sub-árvores fossem conexas então não seriam sub-árvores, pois a DFS pesquisaria até exaustar uma sub-componente.

**Def.** (Aresta para Trás) Dizemos que  $(u, v)$  é uma aresta para trás numa DFS quando tanto  $u$  como  $v$  já foram descobertos,  $u$  é descendente de  $v$  e está a ser tratada a aresta.

Para calcular as arestas para trás, mantemos uma lista dos valores de  $low: \mathcal{V} \rightarrow \mathbb{N}$ , para além da usual lista de valores de descoberta ( $d$ ) e de pais ( $\pi$ ).  $low(v)$  é o mínimo entre o tempo da sua descoberta e o  $low$  dos seus descendentes directos. Isto significa que, se existir uma aresta para trás  $(u, w)$ , onde  $u$  é descendente de  $v$  e  $w$  antecessor, então  $low(u) = low(w)$ . Assim, conseguimos descobrir se um vértice (ou algum dos seus descendentes) tem arestas para um dos antecessores de  $v$  na árvore de DFS. Podemos sintetizar dois predicados, um para dados dois vértices descobrir se existe uma aresta para trás (e subseqüentemente actualizar o  $low$ ) e outro para, dada uma sub-árvore DFS (completamente explorada), descobrir se há uma aresta para trás da raiz da sub-árvore.

$$\begin{aligned} is\_back\_edge(u, w) &\leftarrow \\ d(w) &< d(u) \wedge \pi(u) \neq w \\ subtree\_has\_backedge(v) &\leftarrow \\ \exists c \in Children(v) : low(c) &< low(v) \end{aligned}$$

Com base nisto, podemos verificar se um ponto é de articulação ou não utilizando o seguinte predicado.

$$\begin{aligned} is\_articulation\_point(v) &\leftarrow \\ (is\_dfs\_root(v) \wedge n\_children(v) &\geq 2) \vee \\ (\neg is\_dfs\_root(v) \wedge \exists u \in Children(v) : & \\ \neg subtree\_has\_backedge(u)) & \end{aligned}$$

Fazendo este cálculo para todos os nós, à medida que se faz a DFS (mantendo uma

<sup>2</sup>Depth-First Search, ou DFS, em inglês

estrutura de booleanos que indica se eles são nós ou não), calculando o predicado se o nó ainda não estiver estabelecido como ponto de articulação.

### C. Componente Máxima

Depois de remover os nós (marcando-os com uma cor especial), calculamos uma versão simplificada da DFS, onde mantemos apenas o tamanho da maior componente.

## III. ANÁLISE TEÓRICA

### A. Representação do Grafo

Representamos o grafo por uma lista de adjacências. Notemos que esta representação é  $O(V + E)$  [1] em uso de memória e que a operação de obter a lista e adjacências de um nó é  $O(1)$ .

A nossa implementação tem 4 passos relevantes:

- 1) Construir o grafo;
- 2) Aplicar uma DFS, como descrito em II-B para calcular os pontos de articulação;
- 3) Remover os pontos de articulação;
- 4) Tornar a aplicar uma DFS, como descrito em II-C, para calcular o tamanho da maior sub-rede.

Notemos que, uma vez calculados os pontos de articulação, removê-los é uma operação trivial, completada em  $\Theta(V)$ , no pior caso (basta marcá-los com uma cor especial).

### B. Construção do Grafo

O input é dado em 3 secções: 1) Uma linha com o número de roteadores; 2) Uma linha com o número de ligações; 3) lista de ligações.

Como a adição de um nó no grafo é executada em tempo constante e a adição de uma aresta também, concluímos que a construção do grafo é  $\Theta(V + E)$ .

### C. Pesquisa em Profundidade

Aplicar uma DFS é tem complexidade temporal  $\Theta(V + E)$  [1]. Mostramos agora que a nossas adaptações da pesquisa limitam-se a fazer computações em tempo constante, e que por isso é também um  $\Theta(V + E)$ .

Na primeira aplicação, que calcula os pontos de articulação e os identificadores das sub-redes, são feitas as seguintes operações:

- Inicializações:  $\Theta(1)$
- Obtenção da lista nós adjacentes:  $\Theta(1)$

Para cada adjacente não descoberto

- Incrementação do número de filhos:  $\Theta(1)$
- Inicialização do valor do pai:  $\Theta(1)$
- Actualização do *low* (comparação e eventual atribuição) :  $\Theta(1)$
- Verificação se é ponto de articulação :  $\Theta(1)$ , pois as dependências do predicado apresentado em II-B já se encontram calculadas, ou são calculáveis em  $\Theta(1)$

Para cada adjacente (fora o pai) não descoberto:

- Actualização do *low*:  $\Theta(1)$

Na segunda aplicação a visita é mais simples:

- Inicialização da cor:  $\Theta(1)$
- Obtenção da lista nós adjacentes:  $\Theta(1)$

Para cada adjacente não descoberto

- Incrementação do número de nós da sub-rede:  $\Theta(1)$

Como todas as operações da visita não estruturais à pesquisa em profundidade são  $\Theta(1)$  e a própria pesquisa é  $\Theta(V + E)$ , concluímos que ambas as aplicações têm uma complexidade  $\Theta(V + E)$ .

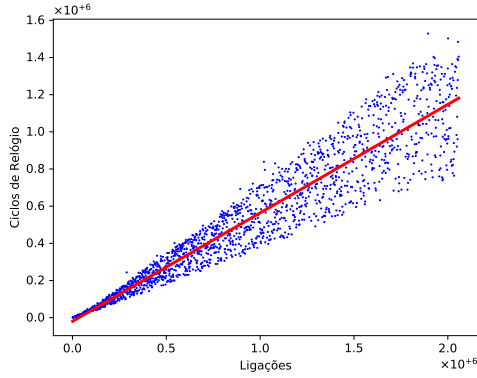


Fig. 1. Resultados experimentais: Latência em função do número de arestas

#### D. Avaliação do Algoritmo Proposto

Concluimos que o algoritmo tem complexidade:

$$\begin{aligned}
 & \Theta(V + E) \\
 & \text{(Construção do Grafo)} \\
 & + \Theta(V + E) \\
 & \text{(Aplicação 1ª DFS)} \\
 & + \Theta(V) \\
 & \text{(Rem. P. de Articulação)} \\
 & + \Theta(V + E) \\
 & \text{(Aplicação 2ª DFS)} \\
 & = 3\Theta(V + E) + \Theta(V) = \Theta(V + E)
 \end{aligned}$$

#### IV. AVALIAÇÃO EXPERIMENTAL

Para a avaliação experimental, geramos testes com as seguintes variantes.

- V: número de roteadores na rede
- E: número de ligações na rede
- S: número de sub-redes

Consideramos mais interessante o caso onde  $E > V$ , pois no caso contrário há pelo menos  $V - (2 * E)$  vértices sozinhos. Assim, há uma maior complexidade nas sub-redes.

Na Figura 1 está exposta a relação entre o número de arestas e a complexidade temporal (medida em número de ciclos de relógio). A recta de regressão linear encontrada tem um coeficiente de determinação  $R^2 = 0.897$ , pelo que concluimos que o desempenho da nossa solução, como previsto

dada a análise teórica, varia linearmente com o número de arestas do grafo.

Os testes foram efectuados numa arquitectura Intel® Core™ i5-6200U. O código foi instrumentado directamente, através da inclusão de duas instruções, uma para obter o valor do relógio no momento inicial e outra no momento final.

#### V. CONCLUSÃO

Apresentamos uma solução para o problema de descoberta de pontos de falha única numa rede, bem como a simulação do pior caso para a rede (a falha simultânea de todos os pontos de falha única). Para tal, mapeamos o problema para um problema de grafos, onde procuramos os pontos de articulação. Para tal, recorremos a uma variante da pesquisa em profundidade. Para calcular o tamanho da maior componente, limitamo-nos a aplicar de novo a pesquisa em profundidade. Mostramos teoricamente e confirmamos experimentalmente esses resultados.

#### REFERÊNCIAS

- [1] T. Cormen, C. Leiserson e L. R. Rivest, *Introduction to Algorithms* 1ª edição. Cambridge, Massachusetts: The MIT Press, 1990.
- [2] R. E. Tarjan. "Depth-First Search and Linear Graph Algorithms" in *SIAM Journal on Computing*, Vol. 1, No. 2, June 1972.
- [3] J. Hopcroft and R. E. Tarjan. "Efficient Algorithms for Graph Manipulation" in *Communications of the ACM*, Vol. 16, No. 6, 1973.