Universidade Federal de Viçosa

INF 330 – Teoria e Modelos de Grafos

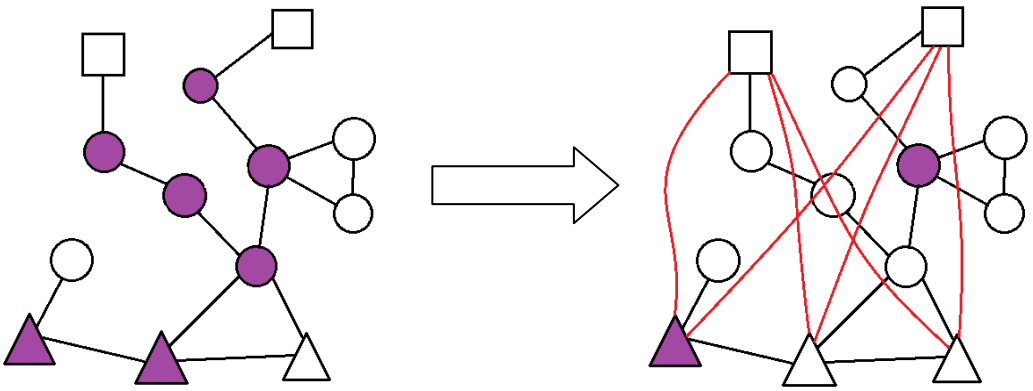
Relatório – Trabalho Prático 1

Professor: Salles Viana Gomes de Magalhães

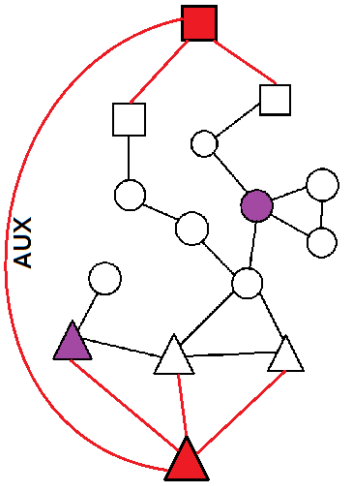
Alunos: Ricson Luiz Oliveira Vilaça – ES-92567
Gabriel Martins Silva – ES-92539
Mayke Daniel de Oliveira Moreira – ES-82873

A teoria

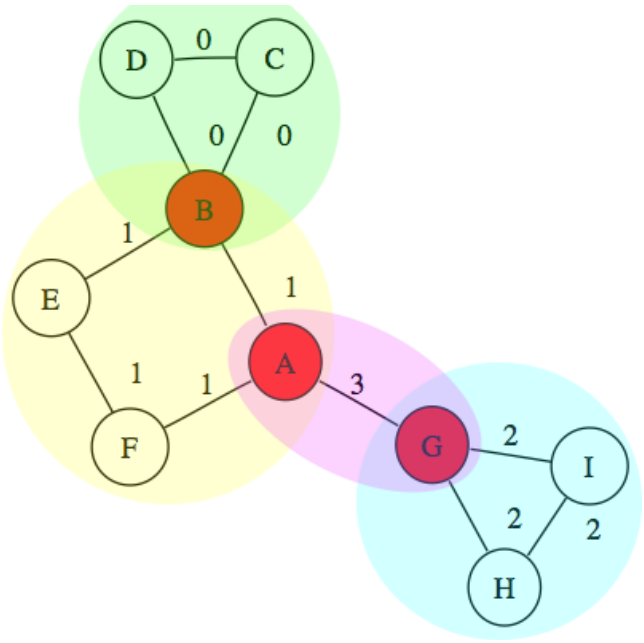
Após algum tempo fazendo pesquisas na parte teórica dos grafos envolvidos no problema, nossa ideia inicial consistia basicamente em, para cada componente conexo do grafo, ligar todos os starting points a todos os controllers e, a partir disso, encontrar todas as articulações dele. Veja abaixo um exemplo (Nas visualizações dos grafos, iremos representar os vértices "starting points" em formato triangular e os "controllers" em formato retangular. Os demais serão circulares. Arestas e vértices "artificiais" serão representados em vermelho e articulações serão representadas em roxo):



Pudemos perceber que a utilização das arestas auxiliares reduziu o número de articulações. Tais arestas garantem que a remoção de nenhum vértice em algum caminho entre algum starting point e algum controller desconectará o subgrafo que é solução do problema. Porém a partir disso não conseguimos encontrar uma forma eficiente de encontrar quais "features" fazem parte da solução. Além disso, consultamos nosso Professor e ele nos alertou que, para casos em que o grafo possuísse um grande número de starting points e controllers, nosso algoritmo teria complexidade de aproximadamente $O(N^2)$, em que N é o número de starting points + controllers. Tal impasse foi resolvido com a criação de dois vértices artificiais para cada componente conexo: Um que ligaria todos starting points e um que ligaria todos os controllers. Assim criaríamos as arestas auxiliares entre tais vértices, como podemos ver na imagem ao lado.

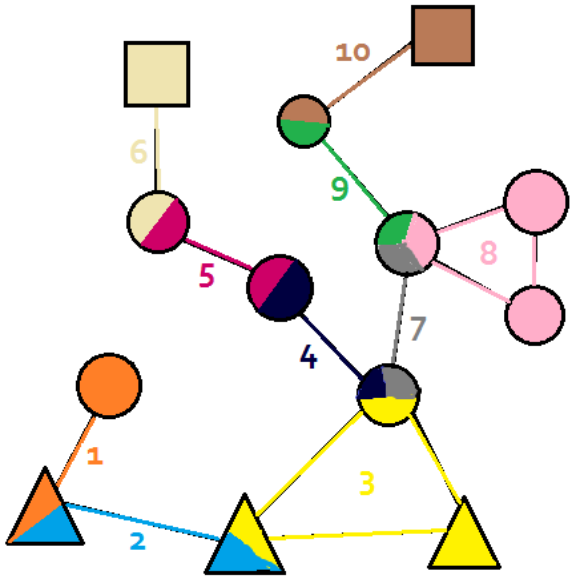


Pesquisando mais a fundo problemas que envolviam articulações, descobrimos um conceito crucial para a resolução do problema: Componentes biconectados. Por definição, um GRAFO biconectado é um grafo conexo que não possui articulações. Os componentes biconectados de um grafo G são subgrafos de G que não possuem articulações:



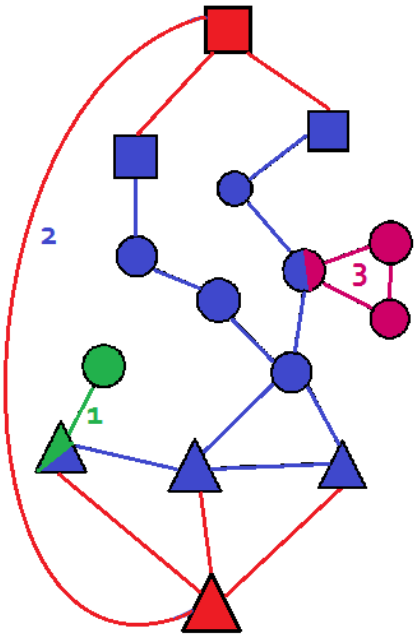
(imagem obtida em https://www.boost.org/doc/libs/1_33_1/libs/graph/doc/biconnected_components.html)

Cada vértice pode pertencer a mais de um componente biconectado, e cada aresta pertence a apenas um. A solução do problema consiste, basicamente, em encontrar todas as arestas e vértices que estão no mesmo componente biconectado das arestas de algum caminho entre cada par de starting point e controller. Observe a divisão de componentes biconectados no grafo inicialmente dado:



Percebe-se que os componentes biconectados 2, 3, 4, 5, 6, 7, 9 e 10 são a solução neste caso, e que basta verificar o componente biconectado das arestas de qualquer caminho entre cada par de starting point e controller e adicioná-lo todo à solução.

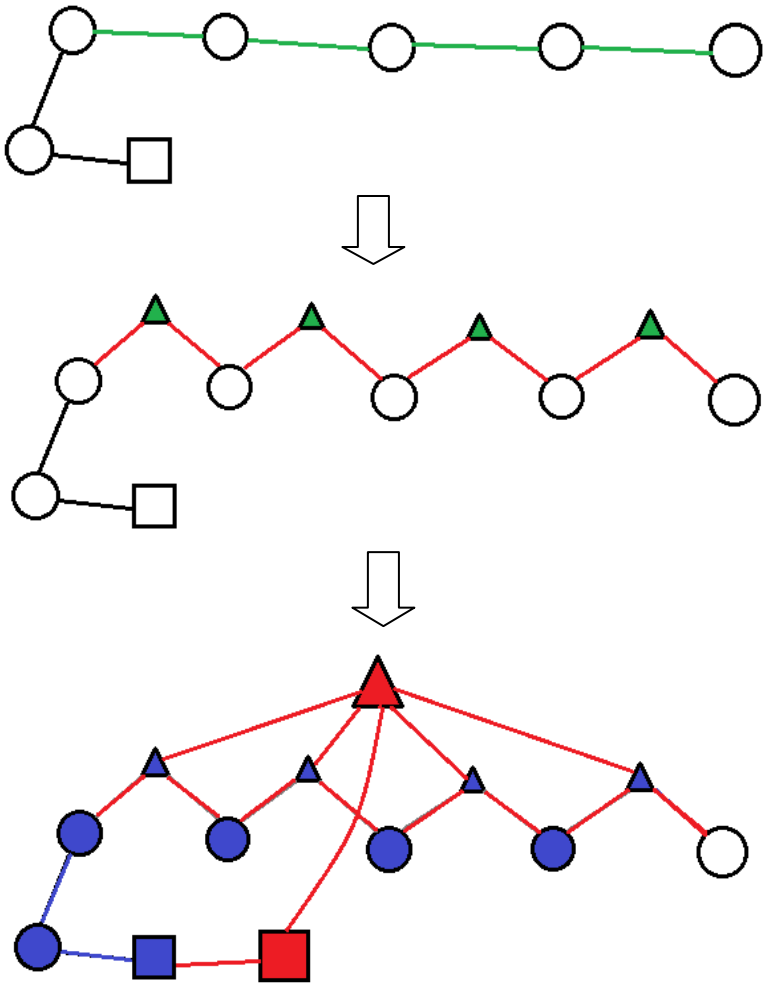
Nesse ponto, percebemos que a utilização das arestas auxiliares inicialmente consideradas facilitaria muito na obtenção dos features desejados, e seria crucial no aumento da eficiência do problema. Veja a divisão de componentes biconectados após a inserção delas:



(A solução será representada em azul)

Agora bastaria a nós apenas verificar quais são os componentes biconectados das arestas auxiliares (no caso, apenas o 2) criadas, assim a solução inclui todas as arestas pertencentes a tais componentes biconectados (exceto as auxiliares criadas) e os dois vértices adjacentes a cada uma delas. É possível perceber também que cada componente conexo do grafo terá no máximo um componente biconectado pertencente à solução, graças à aresta auxiliar existente neles (componentes conexos)

Para tratar casos em que o starting point é uma aresta, consultamos nosso Professor e a sugestão foi simples e eficiente: ao invés de inserir a aresta E entre os vértices U e V, criar um vértice W que representa tal aresta, ligá-lo a U e V e resolver o problema normalmente, pois W estará na solução e terá o mesmo nome da aresta E. Veja um exemplo para varias arestas designadas como starting point (em verde):



A implementação

Para representar o grafo, utilizamos a biblioteca Boost para c++, pois além de armazená-lo e manipulá-lo de forma eficiente, possui já implementadas funções úteis na resolução do problema, principalmente a biconnected_components. Para a leitura do arquivo json, utilizamos a biblioteca Rapidjson que, segundo análises encontradas na internet, é uma das mais eficientes.

Basicamente, convertemos todos os nomes dos vértices e arestas para inteiros, que servem como índices, e inserimos em dicionários de entrada (string --> int) e de saída (int --> string). Considerando que uma via pode conter várias arestas, podem existir arestas com o mesmo nome, portanto no dicionário de entrada para arestas foi necessário utilizar a estrutura de dados "multimap", que suporta vários valores para uma mesma chave (uma string pode ser traduzida para inteiros diferentes). Tendo o grafo já armazenado com seus índices e já tratado o caso de aresta como starting point, criamos as arestas auxiliares (para cada componente conexo, pois não podemos ligar um starting point a um controller que se encontra em outro componente conexo). Feito isso, aplicamos a função que encontra o componente biconectado de cada aresta (complexidade O(V+E), onde V é o número total de vértices e E é o número total de arestas) e armazenamos os componentes biconectados das arestas auxiliares. Então, para finalizar, iteramos sobre todas as arestas do grafo, imprimindo, para cada uma que fizer parte de algum componente biconectado de alguma aresta auxiliar, imprimimos seu nome e o nome de seus vértices adjacentes. Mais detalhes no arquivo "main.cpp".

- Para compilar e executar o nosso programa:

$ g++ main.cpp –O3 –std=c++11

$ time ./a.out "<network.json>" "<startingPoints.txt>" "<output.txt>"

Testes

	Vertices	Arestas	Tempo (s)
SampleDataset1	17	17	0.026
EsriNaperville	8465	9101	0.177
Patherdosreny	500	13000	0.098
Salles(600MB)	100000	1000000	15.803

Fontes de consulta / Bibliotecas utilizadas

Salles Viana Gomes de Magalhães (dicas na resolução e disponibilização de casos teste)

Tcvdijk (casos teste): www.github.com/tcvdijk/wupstream

www.ime.usp.br

www.geeksforgeeks.com.br

www.stackoverflow.com

Boost C++: www.boost.org

RapidJson: www.rapidjson.org