UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

TRABALHO DE CONCLUSÃO DE CURSO

Estudo do problema da bissecção mínima em árvores

Aluna: Karina Suemi Awoki

Orientadora: Cristina Gomes Fernandes

Resumo

Este trabalho de conclusão de curso consistiu no estudo e implementação de um algoritmo linear que, dado uma árvore, produz uma bissecção de largura controlada. Em especial, para árvores de grau limitado por uma constante, o algoritmo é uma aproximação para o problema da bissecção mínima. Este algoritmo faz parte da tese de doutorado de Tina Janne Schmidt, da *Technische Universität München*. O trabalho inclui a análise da correção e de consumo de tempo do algoritmo, que envolve uma série de resultados, bem como um estudo experimental do algoritmo. Para esse estudo, foi implementado também o algoritmo de Jansen, Karpinski, Lingas e Seidel, que produz uma bissecção mínima, especializado para árvores. Além disso, o trabalho inclui ainda resultados de complexidade computacional para o problema da bissecção mínima.

Sumário

Parte I

Parte Objetiva

1 Introdução

O assunto abordado neste trabalho de conclusão de curso (TCC) se enquadra na área de Otimização Combinatória e diz respeito ao *Problema da Bissecção Mínima*. Para definir o problema precisamente, seguem primeiro algumas definições.

Seja $G = (V, E)$ um grafo e B e W conjuntos de vértices de G. Dizemos que (B, W) é um **corte** de G se $B \cup W = V$ e $B \cap W = \emptyset$. Dizemos também que uma aresta e de G está no corte (B, W) se e tem uma ponta em B e outra em W. Denotamos o número de arestas no corte (B, W) por **largura** do corte ou por $e_G(B, W)$. O corte (B, W) é uma **bissecção** se $|B| = |W|$ quando $|V|$ é par, ou se $||B| - |W|| = 1$ quando $|V|$ é ímpar.

O *Problema da Bissecção Mínima* consiste em, dado um grafo, encontrar uma bissecção no grafo de largura mínima.

Sabe-se que o Problema da Bissecção Mínima é NP-difícil [5] e a melhor aproximação conhecida para o caso geral do problema tem razão $O(\lg n)$ [9], onde n é o número de vértices do grafo. Por outro lado, sabe-se que, para árvores e para grafos que se assemelham a árvores, há um algoritmo polinomial para encontrar uma bissecção mínima, proposto por Jansen, Karpinski, Lingas e Seidel [6].

Um melhor entendimento da estrutura dos grafos cuja bissecção mínima tem largura grande pode ajudar no desenvolvimento de bons algoritmos de aproximação para o problema ou ajudar na identificação de classes de grafos onde o problema torna-se especialmente difícil. Para tanto, certas questões têm sido estudadas em árvores, em grafos que têm uma estrutura semelhante às árvores, e também em grafos planares [3, 4], para os quais a complexidade do problema encontra-se em aberto.

Vários outros resultados são conhecidos para o Problema da Bissecção Mínima, porém este TCC se concentrou no estudo do problema em árvores.

1.1 Objetivos

O principal objetivo deste trabalho é o estudo, implementação e análise de um algoritmo recente, proposto por Fernandes, Schmidt e Taraz [3], para

encontrar uma bissecção aproximadamente mínima em árvores de grau limitado. Denotaremos esse algoritmo por FST.

A vantagem do algoritmo FST sobre o algoritmo de Jansen et al. [6], que encontra uma bissecção de largura mínima, é que este tem um consumo de tempo linear, enquanto que o segundo tem um consumo de tempo cúbico no número de vértices da árvore. Implementamos também o algoritmo de Jansen et al., para fins de comparação da largura das bissecções produzidas pelo algoritmo FST.

O algoritmo de Jansen et al. baseia-se em programação dinâmica. Já o algoritmo FST é mais refinado. Para atingir um consumo de tempo linear, ele utiliza técnicas bem conhecidas de percursos de árvore, como busca em profundidade, bem como um rastreamento de informações mais cuidadoso que permite que o processamento todo seja executado em tempo linear.

Fizemos uma análise da performance do algoritmo FST em árvores binárias geradas aleatoriamente, em árvores ternárias completas e, em árvores de caminho longo. Também obtivemos instâncias reais de um problema relacionado e as adaptamos para utilizá-las no nosso estudo experimental.

1.2 Método

O estudo do algoritmo FST seguiu um roteiro de como a implementação foi feita, conforme a proposta deste TCC. A linguagem escolhida para a implementação foi `C` e os algoritmos implementados foram colocados na página do gitHub em `https://github.com/karinaawoki/TCC` e `https://github.com/karinaawoki/jansen_arvores`. O roteiro consistiu em uma divisão da implementação do algoritmo em várias etapas, que permitiu um melhor acompanhamento do trabalho desenvolvido, e uma organização da forma de estudo. O algoritmo encontra-se descrito em um documento longo [10] juntamente com a sua análise teórica. Este documento serviu como base para o entendimento de cada etapa, e de como a implementação de cada etapa deveria ser feita para que o consumo de tempo fosse de fato linear.

Reuniões frequentes foram feitas junto à supervisora para apresentar as

etapas já implementadas, bem como para tirar dúvidas sobre as próximas etapas ou detalhes da implementação. Em maio e junho, Tina Schmidt, uma das autoras do algoritmo que foi implementado, visitou o IME-USP, e a parte da implementação que estava pronta foi apresentada a ela, e algumas discussões foram feitas. Em especial, a Tina conseguiu um conjunto de instâncias reais com Steffen Borgwardt, feitas por Kampeier [7], para usarmos em nossos experimentos.

Em paralelo ao estudo e desenvolvimento da implementação, foram estudados também dois artigos da literatura: o artigo de Garey, Johnson e Stockmeyer [5] que contém a prova de que o Problema da Bissecção Mínima é NP-difícil, e o artigo de Jansen et al. [6] que apresenta o algoritmo que resolve o problema em árvores. Após a implementação do algoritmo FST, foi feita a implementação do algoritmo de Jansen et al. e o estudo experimental do algoritmo FST.

2 Definições

Para um grafo G, denotamos seu conjunto de vértices por $V(G)$ e seu conjunto de arestas por $E(G)$.

O número de arestas que incidem em um determinado vértice v de G é chamado de **grau** de v em G e é representado por $\text{grau}_G(v)$. O **grau máximo**, que é o grau de um vértice de maior grau em G, é representado por $\Delta(G)$.

Um **caminho** em G é uma sequência de vértices $v_0, v_1, \ldots, v_{p-1}, v_p$ onde v_{k-1}, v_k é uma aresta de G para todo $k = 1, 2, \ldots, p$. O número p é o **comprimento** do caminho.

Um **caminho máximo** em G é um caminho em G de comprimento máximo. O comprimento de um caminho máximo em G é o **diâmetro** de G, e é denotado por $\text{diam}(G)$.

Seja G um grafo não necessariamente conexo. O **diâmetro relativo** de G é a razão entre o número de vértices de um caminho máximo de G e o

número total de vértices de G. Este pode também ser representado por

$$\text{diam}^*(G) = \frac{\sum_{G' \text{ componente conexa de } G} (\text{diam}(G') + 1)}{|V(G)|}.$$

Note que $0 < \text{diam}^*(G) \leq 1$ para todo grafo G com $V(G) \neq \emptyset$, pois há pelo menos um vértice de G num caminho máximo de G, e no máximo todos os vértices de G.

Em uma árvore T, existe um único caminho que conecta quaisquer dois vértices x e y. Chamaremos tal caminho em T de x, y–**caminho**.

A **distância** entre dois vértices x e y de T é representada por $\text{dist}(x, y)$, e é o comprimento do $x, y-$caminho em T.

Denotamos por $[n]$ o conjunto $\{1, 2, \ldots, n\}$.

3 Caminho máximo em árvores

Um dos passos do algoritmo FST é a determinação de um caminho máximo em uma árvore. Encontrar um caminho máximo em uma árvore $T = (V, E)$ é uma tarefa computacionalmente simples que pode ser realizada em tempo $O(n)$, sendo n o número de vértices de T, ou seja, $n = |V|$.

Primeiramente escolhe-se um vértice arbitrário $v \in V$ e encontra-se um vértice y_0 mais distante de v em T. Depois, repetimos o mesmo processo a partir de y_0, encontrando um vértice x_0 mais distante de y_0 em T.

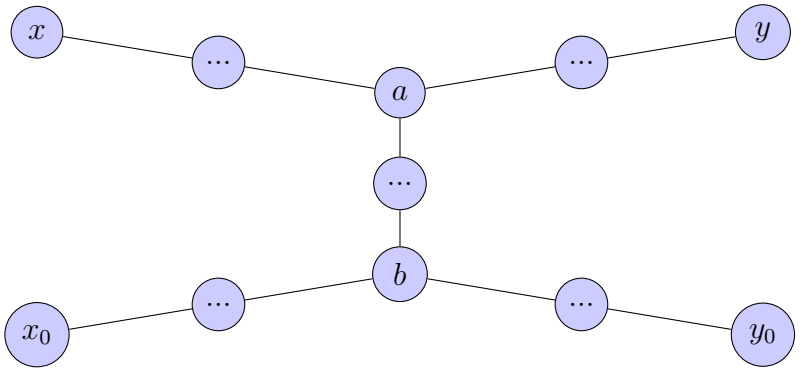
Para encontrar um vértice mais distante de algum vértice dado, basta usar uma busca em largura em T. Feito isso, como demonstraremos a seguir, temos um caminho máximo em T: o x_0, y_0-caminho.

Lema 1. *O* x_0, y_0-caminho *é um caminho máximo em* T.

Demonstração. Suponha que exista um outro caminho mais longo que o x_0, y_0-caminho e sejam x e y os seus extremos.

Caso 1: O $x, y-$caminho e o x_0, y_0-caminho não possuem vértices em comum.

Existe um $a, b-$caminho em T, de comprimento $c \geq 1$, que conecta os dois caminhos, com apenas o vértice a no $x, y-$caminho e apenas o vértice b no x_0, y_0-caminho.

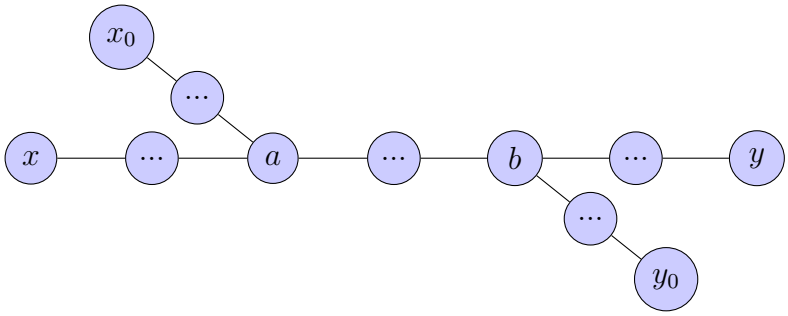


Como estamos em uma árvore, $\text{dist}(x, y) = \text{dist}(x, a) + \text{dist}(a, y)$ e $\text{dist}(x_0, y_0) = \text{dist}(x_0, b) + \text{dist}(b, y_0)$. Como o $x, y-$caminho é máximo, $\text{dist}(x, y) \geq \text{dist}(x_0, y)$, e isso implica que $\text{dist}(x, a) \geq \text{dist}(x_0, b) + c$.

Temos que $\text{dist}(x_0, y_0) \geq \text{dist}(x, y_0)$, pois x_0 é um vértice mais distante de y_0 e isso implica que $\text{dist}(x_0, b) \geq \text{dist}(x, a) + c$. Logo temos que $\text{dist}(x, a) \geq \text{dist}(x, a) + 2c$, uma contradição, visto que $c \geq 1$.

Caso 2: O $x, y-$caminho e o x_0, y_0-caminho possuem vértices em comum.

A interseção entre esses dois caminhos é um $a, b-$caminho de comprimento $c \geq 0$. Podemos assumir, sem perda de generalidade, que $\text{dist}(x, a) \leq \text{dist}(x, b)$ e que $\text{dist}(x_0, a) \leq \text{dist}(x_0, b)$, como na figura seguinte. (Do contrário troque a por b e possivelmente x por y.)



Temos que $\text{dist}(x_0, a) \geq \text{dist}(x, a)$, pois x_0 é um vértice mais distante de y_0. Por outro lado, como x é um vértice mais distante de y, temos também que $\text{dist}(x, a) \geq \text{dist}(x_0, a)$ e, portanto, $\text{dist}(x_0, a) = \text{dist}(x, a)$. Similarmente, como y é um vértice mais distante de x, vale que $\text{dist}(y, b) \geq \text{dist}(y_0, b)$.

Agora consideremos o vértice v que foi escolhido arbitrariamente no início do algoritmo, e seja v' o vértice mais próximo de v no x_0, y_0-caminho. Como y_0 é um vértice mais distante de v,

$$\begin{aligned} \text{dist}(v', y_0) &\geq \text{dist}(v', y) = \text{dist}(v', b) + \text{dist}(b, y) \\ &\geq \text{dist}(v', b) + \text{dist}(b, y_0) \geq \text{dist}(v', y_0), \end{aligned}$$

o que implica que $\text{dist}(v', y_0) = \text{dist}(v', y)$ e $\text{dist}(b, y) = \text{dist}(b, y_0)$. Ou seja, $\text{dist}(x, y) = \text{dist}(x_0, y_0)$, completando a prova. □

4 Lemas de cortes aproximados

Em [10], são apresentados algoritmos, que, dados uma floresta G e um inteiro $0 < m \leq n$, onde n é o número de vértices de G, produzem cortes com algumas propriedades. Os algoritmos em questão são usados como subrotinas no algoritmo FST.

Abaixo reproduzimos os três resultados de [10] que atestam as propriedades dos cortes produzidos por estes algoritmos. O primeiro deles é o mais simples e o algoritmo vinculado a ele é usado no segundo.

Lema 2. *Para toda árvore T com n vértices e todo $m \in [n]$, existe um corte (B, W) em T tal que $\frac{m}{2} < |B| \leq m$ e $e_T(B, W) \leq \Delta(T)$. Um corte que satisfaz esses requisitos pode ser computado em tempo $O(n)$.*

Seja T uma árvore com n vértices e r um vértice de T. Assumiremos que a função NÚMERO_DE_DESCENDENTES(T, r) devolve um vetor que associa cada vértice de T ao número de descendentes desse vértice na árvore T, enraizada em r. Note que essa função pode ser implementada usando-se uma busca em profundidade a partir de r, resultando em um consumo de tempo $O(n)$.

Segue um algoritmo que encontra um corte com as propriedades descritas no lema.

Algoritmo 1: Computa corte aproximado em uma árvore

entrada: árvore $T = (V, E)$ com n vértices, $m \in [n]$ e $r \in V$
saída : corte (B, W) em T com $\frac{m}{2} < |B| \leq m$ e $e_T(B, W) \leq \Delta(T)$

```
1  if m = n then return (V, ∅) ;
2  d ← NÚMERO_DE_DESCENDENTES(T, r);
3  v ← r;
4  while existe filho u de v com d[u] > m do v ← u ;
5  if existe filho u de v com d[u] > m/2 then
6  |  B ← V(T_u);   // T_u é a sub-árvore de T enraizada em u
7  else
8  |  B ← ∅;
9  |  for cada filho u de v do
10 |  |  if |B| + |T_u| ≤ m then
11 |  |  |  B ← B ∪ V(T_u);
12 |  |  else
13 |  |  |  break;
14 |  |  end
15 |  end
16 end
17 return (B, V \ B)
```

Demonstração. Suponha que $T = (V, E)$ e $n = |V|$. Se o algoritmo termina na linha 1, então o corte devolvido satisfaz as condições do lema, dado que $e_G(V, \emptyset) = 0$, e o consumo de tempo é claramente $O(n)$.

Caso contrário, observe que o algoritmo termina de executar o laço da linha 4, pois a árvore é finita, e o faz em tempo $O(n)$ já que cada vértice é percorrido no máximo uma vez.

Seja v o vértice em que o laço da linha 4 termina. Se o algoritmo executa a linha 6, o consumo de tempo é $O(n)$ e o corte (B, W) devolvido é tal que $e_T(B, W) = 1$ e $\frac{m}{2} < |B| \leq m$, pois o filho de v encontrado tem no máximo m descendentes. Se isso não ocorrer, significa que o número de descendentes de todos os filhos de v é no máximo metade de m. Sabe-se que a união de dois conjuntos com no máximo $\frac{m}{2}$ vértices resulta em um conjunto com no máximo m vértices. Portanto, como v possui no mínimo m descendentes e cada filho de v possui no máximo $\frac{m}{2}$ descendentes, ao final do laço das linhas 9-15, $\frac{m}{2} < |B| \leq m$. Desta forma, o algoritmo devolve um corte (B, W) com $\frac{m}{2} < |B| \leq m$ e $e_T(B, W) \leq \Delta(T)$.

O tempo consumido no laço das linhas 9-15 é $O(n)$, pois os filhos de v e seus descendentes são percorridos no máximo uma vez nas iterações do laço e na união de conjuntos da linha 11, respectivamente. □

O lema abaixo representa uma adaptação do algoritmo anterior para florestas em vez de árvores. Note que a adaptação não recebe uma raiz como parâmetro, mas apenas G e m.

Lema 3 ([10, Lema 2])**.** *Para toda floresta G com n vértices e todo $m \in [n]$, existe um corte (B, W) em G tal que $\frac{m}{2} < |B| \leq m$ e $e_G(B, W) \leq \Delta(G)$. Um corte que satisfaz esses requisitos pode ser computado em tempo $O(n)$.*

O algoritmo abaixo determina um corte como o descrito pelo lema.

Algoritmo 2: Computa corte aproximado simples em uma floresta

entrada: floresta $G = (V, E)$ com n vértices e $m \in [n]$
saída : corte (B, W) em G tal que $\frac{m}{2} \leq |B| \leq m$ e $e_G(B, W) \leq \Delta(G)$

```
1  Sejam V_1, V_2, ..., V_k os conjuntos de vértices das componentes de G;
2  B ← ∅;
3  for i = 1 → k do
4      if |B| + |V_i| ≤ m then
5          B ← B ∪ V_i;
6      end
7      else if |B| < m then
8          r ← vértice arbitrário de V_i;
9          (B', W') ← Algoritmo1(G[V_i], m − |B|, r);
10         B ← B ∪ B';
11         break;
12     end
13 end
14 return (B, V \ B);
```

Demonstração. Claro que $|B| \leq m$ após cada execução da linha 5. Ademais, caso a linha 9 seja executada, B' é tal que $\frac{m-|B|}{2} < |B'| \leq m - |B|$ pela descrição do Algoritmo 1. Portanto $|B| \leq m$ após a linha 10 e, como $|B| + \frac{m-|B|}{2} = \frac{m+|B|}{2}$, vale que $|B| > \frac{m}{2}$ após a linha 10 também. Ou seja, na linha 14, temos que $\frac{m}{2} < |B| \leq m$. Além disso, $e_G(B, W) \leq \Delta(G)$ se a linha 10 é executada e $e_G(B, W) = 0$ caso contrário.

Note que, como a linha 1 pode ser feita usando uma busca em profundidade e as demais linhas envolvem verificar o tamanho das componentes (que já foi calculado na linha 1) e uma chamada única ao Algoritmo 1, então essa rotina consome tempo $O(n)$, dado que cada uma das operações citadas são executadas em tempo $O(n)$. □

Finalmente apresentamos o algoritmo que é usado repetidas vezes no algoritmo FST. Ele utiliza o algoritmo anterior como subrotina.

Lema 4 ([10, Lema 3])**.** *Para toda floresta G com n vértices, todo $m \in [n]$ e todo $c \in [0, 1)$, existe um corte (B, W) em G tal que $cm \leq |B| \leq m$ e $e_G(B, W) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$. Um corte que satisfaz esses requisitos pode ser computado em tempo $O\left(\left\lceil \frac{2c}{1-c} \right\rceil n + 1\right)$.*

Segue o algoritmo que encontra um corte com as propriedades do Lema 4.

Algoritmo 3: Computa corte aproximado em uma floresta

entrada: floresta $G = (V, E)$ com n vértices, $m \in [n]$ e $c \in [0, 1)$
saída : corte (B, W) em G tal que $cm \leq |B| \leq m$ e $e_G(B, W) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$

```
1 if c = 0 then return (∅, V);
2 if c ≤ 1/2 then return Algoritmo2(G, m);
3 B ← ∅;
4 while |B| < cm do
5 |   (B', W') ← Algoritmo2(G[V \ B], m − |B|) ;
6 |   B ← B ∪ B';
7 end
8 return (B, V \ B);
```

Demonstração. Se $c = 0$, o Algoritmo 3 devolverá um corte onde $B = \emptyset$ e $e_G(B, W) = 0$. Isso satisfaz as propriedades do Lema 4, dado que $cm = 0 = |B|$.

Se $c \leq \frac{1}{2}$, o algoritmo devolve o corte produzido pelo Algoritmo 2, ou seja, um corte (B', W') tal que $|B'| > \frac{m}{2} \geq cm$ e $e_G(B', W') \leq \Delta(G)$.

Por outro lado, se $c > \frac{1}{2}$, então o conjunto B é inicializado com $\emptyset$. O Algoritmo 2 é executado diversas vezes para encontrar a cada vez um corte (B', W') em $G[V \setminus B]$ tal que $\frac{m - |B|}{2} < |B'| \leq m - |B|$ e $e_{G[V \setminus B]}(B', W') \leq \Delta(G[V \setminus B])$, acrescentando o conjunto B' a B em cada uma das chamadas ao Algoritmo 2. Isso é feito até que $|B| \geq cm$. Note

que em algum momento $|B| \geq cm$, pois a cada vez que executamos o Algoritmo 2, acrescentamos pelo menos um vértice em B. Ademais, $|B| \leq m$ durante todo o processo, já que o conjunto B' obtido pelo Algoritmo 2 é tal que $|B'| \leq m - |B|$. Logo, serão adicionados no máximo $m - |B|$ vértices em B na linha 6.

Verificaremos agora se o consumo de tempo e a largura do corte devolvido pelo Algoritmo 3 satisfazem o que foi proposto no Lema 4.

Se $c = 0$, a largura do corte devolvido é zero, dado que todos os vértices de G estão em W. Portanto $e_G(B, W) = 0 = \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$. Já em relação ao tempo, neste caso o Algoritmo 3 executa apenas operações que levam tempo constante, logo ele leva tempo $O(1)$. Como $\left\lceil \frac{2c}{1-c} \right\rceil n + 1 = 1$, as propriedades do Lema 4 são satisfeitas.

Se $0 < c \leq \frac{1}{2}$, executamos o Algoritmo 2 uma única vez, portanto o tempo é $O(n)$, que equivale a $O\Big(\left\lceil \frac{2c}{1-c} \right\rceil n + 1\Big)$. Em relação à largura do corte, será devolvido o corte (B, W) do Algoritmo 2 sem nenhuma modificação, então sabe-se que $e_G(B, W) \leq \Delta(G)$. Como $0 < c \leq \frac{1}{2}$, temos que $\frac{2c}{1-c} > 0$, que implica que $\left\lceil \frac{2c}{1-c} \right\rceil \geq 1$. Logo, $e_G(B, W) \leq \Delta(G) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$, satisfazendo assim as propriedades do Lema 4.

Caso contrário, $c > \frac{1}{2}$. Como em todas as vezes que executamos a linha 4 do Algoritmo 3, vale que $|B| < cm$, então também é válido que $m - |B| > (1-c)m$. Como $m - |B|$ é o valor do m que passamos para o Algoritmo 2, sabemos que o conjunto B' do corte devolvido terá mais que $\frac{m - |B|}{2} > \frac{(1-c)m}{2}$ vértices. Portanto, executaremos o Algoritmo 2 no máximo $\left\lceil \frac{cm}{\frac{(1-c)m}{2}} \right\rceil = \left\lceil \frac{2c}{1-c} \right\rceil$ vezes, consumindo, no total, tempo $O\Big(\left\lceil \frac{2c}{1-c} \right\rceil n\Big) = O\Big(\left\lceil \frac{2c}{1-c} \right\rceil n + 1\Big)$. Usaremos essa mesma linha de pensamento para calcular a largura máxima do corte devolvido na linha 8. Sabemos que o Algoritmo 2 será chamado no máximo $\left\lceil \frac{2c}{1-c} \right\rceil$ vezes e, para cada uma das vezes, este devolve um corte cuja largura máxima é $\Delta(G)$, portanto a largura máxima do corte devolvido é $\left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$. □

5 Rotulação dos vértices da árvore

O algoritmo FST utiliza um rótulo numérico $x \in [n]$ distinto para cada um dos vértices da árvore T, onde n é o número de vértices de T. Esses rótulos devem ter uma característica específica e por isso são calculados de uma maneira particular. Denotaremos o **rótulo** de um vértice v por $\text{rot}(v)$.

Primeiramente, usaremos o método mencionado na Seção 3 para calcular um caminho P, máximo em T. Para todo vértice $v \in V(P)$, seja T_v a árvore de $T - E(P)$ que contém v.

Dessa forma, teremos sub-árvores enraizadas nos vértices do caminho máximo, como mostrado na figura abaixo, onde denotamos os vértices de P por $v_0, v_1, \ldots, v_p$.

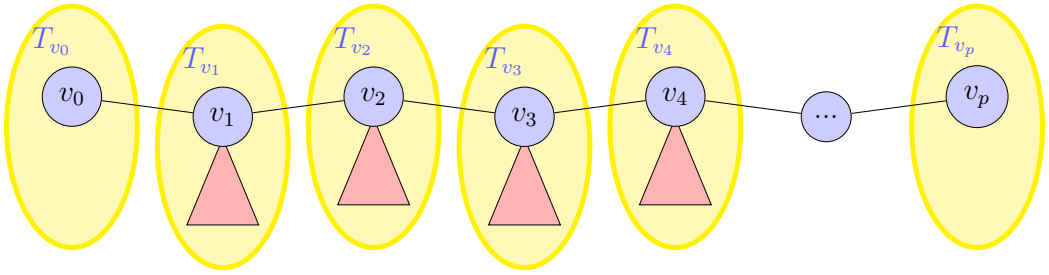


Figura 1: Sub-árvores enraizadas no caminho máximo.

Note que as árvores T_{v_0} e T_{v_p} possuem um único vértice cada uma. Isso ocorre porque v_0 e v_p são extremos do caminho máximo P, e se houvesse mais algum vértice em uma dessas sub-árvores, o caminho P poderia ser estendido, o que levaria a uma contradição.

Cada um dos vértices da árvore T recebe um rótulo de forma que $\text{rot}(v_{i-1}) < \text{rot}(v') \leq \text{rot}(v_i)$ para todo vértice v' da sub-árvore T_{v_i} para $i = 1, 2, \ldots, p$.

5.1 Descrição do algoritmo de rotulação

Essa rotulação pode ser obtida facilmente manipulando as listas de adjacências de T e em seguida, executando uma busca em profundidade. Conforme descrevemos a seguir, ambas as ações consomem tempo linear em n, então pode-se rotular uma árvore com n vértices em tempo $O(n)$.

Para cada vértice v_i contido no caminho P, com $i > 0$, percorremos a sua lista de adjacências e, quando encontrarmos o vértice v_{i-1}, o colocamos no início da lista. Dessa forma, ao aplicarmos uma busca em profundidade nessa árvore usando o vértice v_p como raiz, a busca descerá primeiro pelos vértices do caminho máximo. Os rótulos são atribuídos em pós-ordem, ou seja, a busca atribuirá um rótulo a um vértice do caminho somente quando terminar de rotular os vértices anteriores do caminho e todos os demais vértices da sub-árvore do vértice, garantindo assim que $\mathrm{rot}(v_{i-1}) < \mathrm{rot}(j) \leq \mathrm{rot}(v_i)$ para todo $i \in \{1, \ldots, p\}$ e todo j em $V(T_{v_i})$.

5.2 Mapeamento para frente ou para trás

Vamos definir agora as funções utilizadas pelo algoritmo FST que mapeiam um vértice em outro utilizando a rotulação acima como base. Chamaremos de map_m^+ a função que devolve o m-ésimo próximo vértice na rotulação, considerando que os rótulos estão dispostos de forma circular e, de map_m^-, a função inversa da primeira. Ou seja, map_m^- devolve o m-ésimo vértice anterior na rotulação.

Para facilitar, identificaremos os vértices pelos seus rótulos.

Note que, para todo $m \in [n]$, se existem dois vértices v e v', ambos no caminho P, tais que $v' = \mathrm{map}_m^+(v)$, então o corte (B, W) em T com $B = \{v+1, v+2, \ldots, v+m\}$ é tal que $|B| = m$ e $e_T(B, W) \leq \min\{2, \Delta(T)\} \leq \Delta(T)$. Isso pode ser visto na figura abaixo.

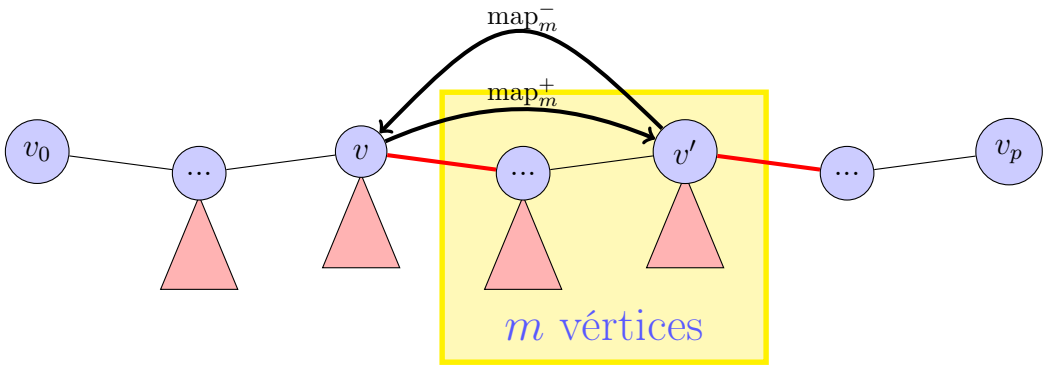


Figura 2: Mapeamento entre os vértices v e v' do caminho máximo induz um corte (B, W) com $|B| = m$ e $e_T(B, W) = 2$.

5.3 Árvores com diâmetro grande

Mais adiante usaremos o lema abaixo na prova de um dos teoremas.

Lema 5 ([10, Lema 5])**.** *Para toda árvore T com n vértices e* $\text{diam}^*(T) > \frac{1}{2}$ *e para todo $m \in [n]$, existe um corte (B, W) em T com $|B| = m$ e $e_T(B, W) \leq 2$. Um corte que satisfaz esses requisitos pode ser computado em tempo $O(n)$.*

Demonstração. Considere um caminho máximo P em T e a rotulação dos vértices de T induzida por ele. Podemos ver que a função $\text{map}_m^+(\cdot)$ é bijetora, sendo $\text{map}_m^-(\cdot)$ sua função inversa.

Em uma árvore T com $\text{diam}^*(T) > \frac{1}{2}$, um caminho máximo contém mais que a metade dos vértices da árvore, logo existem mais vértices no caminho P do que fora dele.

Como mostrado anteriormente, se um vértice v é mapeado em um vértice v' e ambos estão no caminho máximo P, então existe um corte (B, W) em T com $e_T(B, W) \leq 2$. Dado que $\text{diam}^*(T) > \frac{1}{2}$, e usando o fato de que $\text{map}_m^+(\cdot)$ é uma função bijetora e que existem mais vértices no caminho máximo P do que fora dele, temos que pelo menos um vértice de P

será mapeado em outro vértice de P. Assim, obtemos um corte (B, W), com $e_T(B, W) \leq 2$, como descrito na subseção anterior.

Um algoritmo que encontra um corte desse tipo, em árvores com essa propriedade, percorre os vértices do caminho máximo P e verifica se algum deles é mapeado em um vértice de P. Isso pode ser feito em tempo $O(n)$ se construirmos um vetor binário que indica se cada um dos vértices de T está ou não no caminho máximo P. □

6 Dobra do diâmetro relativo

O teorema a seguir é o coração do algoritmo FST. Ele é chamado de *teorema da dobra do diâmetro relativo*. Nele utilizamos a seguinte notação para uma partição (X, Y, Z) de $V(T)$. Denotamos por $e_T(X, Y, Z)$ o número de arestas de T cujos extremos estão em conjuntos distintos da partição (X, Y, Z).

Teorema 1 ([10, Teorema 4])**.** *Para toda árvore T com n vértices e todo $m \in [n]$, o conjunto de vértices de T pode ser particionado em três partes B, W e S tais que vale um dos seguintes itens:*

1. $|B| = m$, $S = \emptyset$, *e* $e_T(B, W) \leq 2$, *ou*
2. $|B| \leq m \leq |B| + |S|$, *com* $0 < |S| \leq \dfrac{n}{2}$, $e_T(B, W, S) \leq \dfrac{2 \cdot \Delta(T)}{\operatorname{diam}^*(T)}$, *e* $\operatorname{diam}^*(T[S]) \geq 2 \operatorname{diam}^*(T)$.

A prova deste teorema usa os seguintes conceitos. Para os vértices $a, b, c \in V(T)$, dizemos que b **está entre** a e c se $a = b$ ou $b = c$ ou se chegamos em b antes de c quando aumentamos o valor de a de um em um. Dados vértices $x, z \in V(P)$, dizemos que se $x + 1 \in T_z$, então o vértice z **vem depois de** x **em** P e x **vem antes de** z **em** P.

Um vértice $z \in V(P)$ é **b-especial** se existe um vértice $x \in T_z$ tal que $\text{map}_m^-(x) \in V(P)$, e z é **f-especial** se existe um vértice $x \in T_z$ tal que $\text{map}_m^+(x) \in V(P)$. (O b vem de *backward* e o f de *forward*.)

Demonstração. Considere uma árvore T com n vértices e um inteiro $m \in [n]$. Seja P um caminho máximo em T e, considere os vértices rotulados como descrito na Seção 5. Obteremos um corte (B, W, S) da seguinte forma:

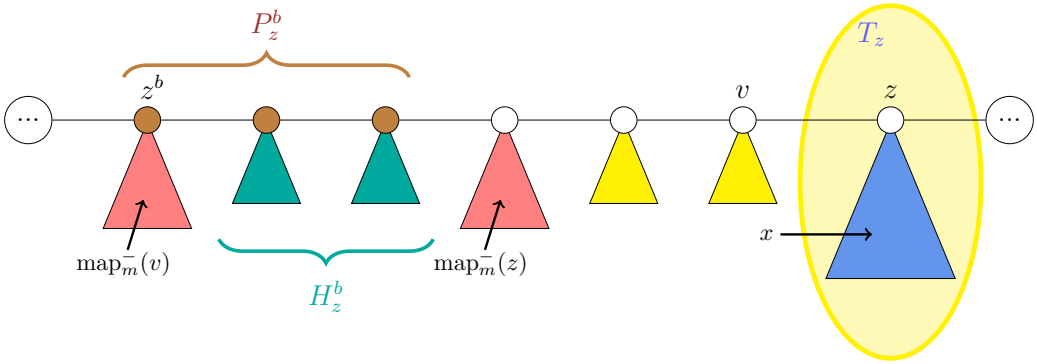
Caso 1: Existe um vértice $v \in V(P)$ tal que $\text{map}_m^+(v) \in V(P)$.

A partição (B, W, S) com $B = \{v+1, v+2, \ldots,\ v+m\}$, $W = V(T) \setminus B$ e $S = \emptyset$ satisfaz as propriedades do primeiro caso do teorema, como explicado na Seção 5.2.

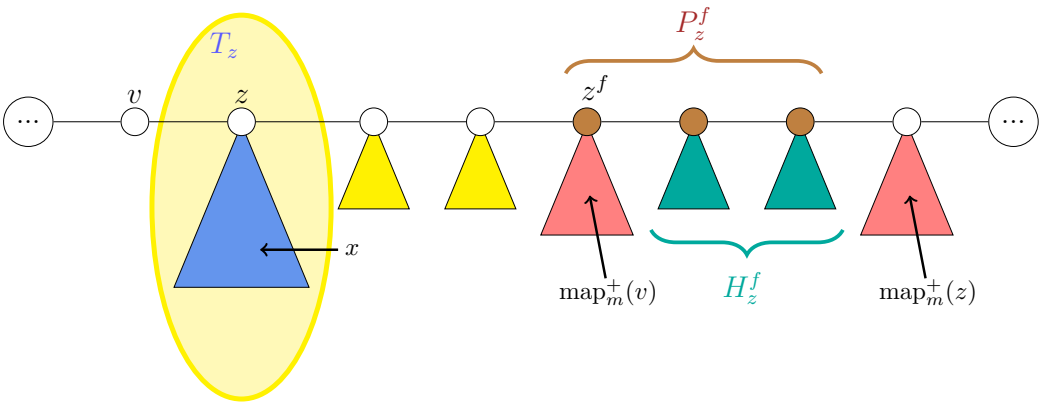
Caso 2: Para todo vértice $v \in V(P)$, $\text{map}_m^+(v) \notin V(P)$.

Como $\text{map}_m^+(\cdot)$ e $\text{map}_m^-(\cdot)$ são funções bijetoras, existem $|V(P)|$ ou mais vértices fora de P. Isso implica que $\text{diam}^*(T) \leq \dfrac{n}{2}$, pois no máximo $\dfrac{n}{2}$ vértices estarão em P. Nesse caso, teremos uma partição (B, W, S) que satisfaz as propriedades do segundo caso do teorema. Isso será detalhado melhor a seguir.

Sejam $z \in V(P)$ um vértice b-especial e x o menor vértice de T_z tal que $\text{map}_m^-(x) \in V(P)$. Sejam $z^b = \text{map}_m^-(x)$, $P_z^b = \text{map}_m^-(V(T_z)) \cap V(P)$ e $H_z^f := \cup\{V(T_u) : u \in P_z^b,\ u \neq z^b\}$. Veja a próxima figura com a representação da disposição destes conjuntos de vértices.



De modo análogo, para um vértice $z \in V(P)$ tal que z é f-especial, definimos $z^f = \text{map}_m^+(x)$, onde x é o menor vértice de T_z tal que $\text{map}_m^+(x) \in V(P)$. Também definimos $P_z^f = \text{map}_m^+(V(T_z)) \cap V(P)$ e $H_z^f := \cup\{V(T_u) : u \in P_z^f,\ u \neq z^f\}$. Veja a figura abaixo com a representação da disposição destes conjuntos de vértices.



Note que, no Caso 2, os vértices $\text{map}_m^-(v)$, $\text{map}_m^-(z)$, $\text{map}_m^+(v)$ e $\text{map}_m^+(z)$ não estão em $V(P)$, pois nenhum vértice de P é mapeado em outro vértice de P.

Para facilitar a descrição, adicionamos uma aresta ligando o primeiro ao último vértice de P. Assim os elementos de P_z^b e P_z^f induzem um caminho. Para valores diferentes de z, os conjuntos P_z^b são disjuntos. O mesmo vale para os conjuntos P_z^f.

Usaremos a frente as seguintes propriedades:

1. Um vértice $z \in V(P)$ é b-especial se e somente se $z = u^f$ para um vértice f-especial $u \in V(P)$.

   Demonstração. De fato, se z é b-especial, então existe pelo menos um vértice $v \in V(P)$ tal que $\text{map}_m^+(v) \in T_z$. Seja v um tal vértice com o maior rótulo e u o vértice que vem depois de v em P. Então $\text{map}_m^-(z) \in T_u$ e o vértice a esquerda de z em P não é mapeado em u, logo u é f-especial e $z = u^f$. Por outro lado, se u é f-especial e v é o vértice que vem antes de u em P, temos que $\text{map}_m^+(v) \notin V(P)$. Logo, $\text{map}_m^+(v)$ está em T_{u^f}. Isso se deve ao fato de que $\text{map}_m^+(v)$ não está na árvore do vértice que vem antes de u^f, pois, se estivesse, u^f seria um outro vértice. Logo, $z = u^f$ é b-especial.

   □

2. Um vértice $z \in V(P)$ é f-especial se e somente se $z = u^b$ para um vértice b-especial $u \in V(P)$.

   Podemos verificar que essa propriedade é verdadeira de forma semelhante à verificação da propriedade anterior.

Seja B_{esp} o conjunto de todos os vértices b-especiais, F_{esp} o conjunto de todos os vértices f-especiais e T'_z o conjunto de vértices $V(T_z) \setminus \{z\}$. Usando as propriedades 1 e 2 respectivamente, temos que

$$\sum_{u \in F_{\text{esp}}} |T'_{u^f}| = \sum_{z \in B_{\text{esp}}} |T'_z| \tag{1}$$

$$\sum_{u \in B_{\text{esp}}} |T'_{u^b}| = \sum_{z \in F_{\text{esp}}} |T'_z|. \tag{2}$$

Note que, para cada vértice b-especial z, todos os vértices de H_z^b e de P_z^b são mapeados em vértices de T_z e $\text{map}_m^-(z) \notin H_z^b \cup P_z^b$. O mesmo ocorre com

cada vértice f-especial. Portanto, temos que

$$|T_z| > |P_z^b| + |H_z^b| \;\Rightarrow\; |T'_z| \geq |P_z^b| + |H_z^b| \tag{3}$$
$$|T_z| > |P_z^f| + |H_z^f| \;\Rightarrow\; |T'_z| \geq |P_z^f| + |H_z^f|. \tag{4}$$

Como os conjuntos P_z^b são dois a dois disjuntos e $P_z^b \cup H_z^b \cup T'_{z^b}$ nada mais é do que o conjunto dos vértices das sub-árvores enraizadas nos vértices de P_z^b, temos que os conjuntos $P_z^b \cup H_z^b \cup T'_{z^b}$ também são dois a dois disjuntos. Como todos os vértices de P são mapeados em alguma sub-árvore, todo vértice de $V(P)$ pertence a algum P_z^b. Logo, todo vértice de $V(T)$ pertence a algum $P_z^b \cup H_z^b \cup T'_{z^b}$. O mesmo é válido para $P_z^f \cup H_z^f \cup T'_{z^f}$. Portanto, para $d = \text{diam}^*(T)$,

$$(1-d)n = |V \setminus V(P)| = \sum_{z \in B_{\text{esp}}} \left(|T'_{z^b}| + |H_z^b| \right) \tag{5}$$
$$(1-d)n = |V \setminus V(P)| = \sum_{z \in F_{\text{esp}}} \left(|T'_{z^f}| + |H_z^f| \right) \tag{6}$$
$$dn = |V(P)| = \sum_{z \in B_{\text{esp}}} |P_z^b| \tag{7}$$
$$dn = |V(P)| = \sum_{z \in F_{\text{esp}}} |P_z^f|. \tag{8}$$

Usando as equações (5), (6), (7) e (8) e, as equações (1) e (2) depois, temos que

$$\begin{aligned} \frac{1-d}{d} &= \frac{\displaystyle\sum_{z \in B_{\text{esp}}} \left(|T'_{z^b}| + |H_z^b| \right) + \sum_{z \in F_{\text{esp}}} \left(|T'_{z^f}| + |H_z^f| \right)}{\displaystyle\sum_{z \in B_{\text{esp}}} |P_z^b| + \sum_{z \in F_{\text{esp}}} |P_z^f|} \\ &= \frac{\displaystyle\sum_{z \in B_{\text{esp}}} \left(|T'_z| + |H_z^b| \right) + \sum_{z \in F_{\text{esp}}} \left(|T'_z| + |H_z^f| \right)}{\displaystyle\sum_{z \in B_{\text{esp}}} |P_z^b| + \sum_{z \in F_{\text{esp}}} |P_z^f|}. \end{aligned}$$

Logo

$$\frac{1-d}{d}\sum_{z\in B_{\text{esp}}}|P_z^b|+\frac{1-d}{d}\sum_{z\in F_{\text{esp}}}|P_z^f|=\sum_{z\in B_{\text{esp}}}\Big(|T_z'|+|H_z^b|\Big)+\sum_{z\in F_{\text{esp}}}\Big(|T_z'|+|H_z^f|\Big),$$

o que nos nos leva a dois casos:

(I) $\displaystyle\sum_{z\in B_{\text{esp}}}\Big(|T_z'|+|H_z^b|\Big)\leq\frac{1-d}{d}\sum_{z\in B_{\text{esp}}}|P_z^b|$ ou

(II) $\displaystyle\sum_{z\in B_{\text{esp}}}\Big(|T_z'|+|H_z^b|\Big)>\frac{1-d}{d}\sum_{z\in B_{\text{esp}}}|P_z^b|\ \Rightarrow$
$\displaystyle\sum_{z\in F_{\text{esp}}}\Big(|T_z'|+|H_z^f|\Big)<\frac{1-d}{d}\sum_{z\in F_{\text{esp}}}|P_z^f|\ \Rightarrow$
$\displaystyle\sum_{z\in F_{\text{esp}}}\Big(|T_z'|+|H_z^f|\Big)\leq\frac{1-d}{d}\sum_{z\in F_{\text{esp}}}|P_z^f|.$

Sabemos que, se (I) ocorrer, há um vértice $z\in B_{\text{esp}}$ tal que $|T_z'|+|H_z^b|\leq\frac{1-d}{d}|P_z^b|$, pois se $|T_z'|+|H_z^b|>\frac{1-d}{d}|P_z^b|$ para todo $z\in B_{\text{esp}}$, então (I) não teria ocorrido. O mesmo ocorre com (II). Portanto, um dos casos a seguir sempre será válido:

(a) existe um vértice $z\in B_{\text{esp}}$ tal que $|T_z'|+|H_z^b|\leq\frac{1-d}{d}|P_z^b|$ ou

(b) existe um vértice $z\in F_{\text{esp}}$ tal que $|T_z'|+|H_z^f|\leq\frac{1-d}{d}|P_z^f|$.

Caso (a): Existe um vértice $z\in B_{\text{esp}}$ tal que $|T_z'|+|H_z^b|\leq\frac{1-d}{d}|P_z^b|$. Disso, $|H_z^b|\leq\Big(\frac{1}{d}-1\Big)|P_z^b|-|T_z'|$. Usando a segunda desigualdade em (3), deduzimos que

$$|H_z^b|+|P_z^b|\leq\frac{1}{2d}|P_z^b|.$$

Construiremos a partição (B,W,S) tomando $S=H_z^b\cup P_z^b$. Sabemos que $S\neq\emptyset$, pois z é b-especial, o que implica que $P_z^b\neq\emptyset$. Dessa forma,

temos que

$$|S| = |H_z^b| + |P_z^b| \leq \frac{1}{2d}|P_z^b| \Rightarrow 2d \leq \frac{1}{|S|}|P_z^b|, \tag{9}$$

ou seja, $\text{diam}^*(T[S]) \geq 2d$.

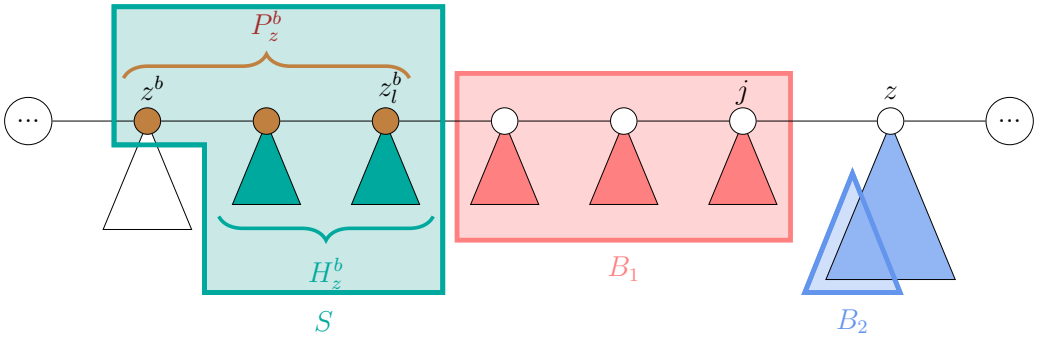
Seja $z_l^b \in P_z^b$ um vértice tal que $\text{map}_m^+(z_l^b)$ é o vértice de T_z com maior rótulo possível, e j o vértice que vem antes de z em P. Seja

$$B_1 = \Big\{v \in V(T) : z_l^b < v \leq j\Big\}.$$

Agora podemos aplicar o Lema 4 à floresta T'_z, com $m - |B_1|$ e $2 - \dfrac{1}{1-d}$ no lugar de m e c, respectivamente. Dessa forma, existe um corte (B_2, W_2) tal que $c \cdot (m - |B_1|) \leq |B_2| \leq m - |B_1|$ e

$$\begin{aligned} e_{T'_z}(B_2, W_2) &\leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(T'_z) = \left\lceil \frac{2}{d} - 4 \right\rceil \Delta(T'_z) \\ &\leq \left\lceil \frac{2}{d} - 4 \right\rceil \Delta(T) \\ &\leq \Big(\frac{2}{d} - 3\Big)\Delta(T). \end{aligned} \tag{10}$$

Como $\text{map}_m^+(z_l^b) \in T'_z$ e todos os vértices entre $\text{map}_1^+(z_l^b)$ e j estão em B_1, sabemos que $m \leq |B_1| + |T'_z|$, que implica que $m - |B_1| < |T_z|$. Logo, $1 \leq m - |B_1| \leq |T'_z|$. Como $d \in (0, \frac{1}{2}]$, temos que $c \in [0, 1)$. Portanto, os valores usados como m e c são compatíveis com o que é pedido no Lema 4. A disposição dos vértices ficará como na figura a seguir.



Temos, então, a partição (B, W, S) com $B = B_1 \cup B_2$ e $W = V(T) \setminus (B \cup S)$. Dado que $\text{map}_m^+(z_l^b) \in T_z$, temos que $m > |B_1|$ e, como obtemos B_2 através do Lema 4 com $m - |B_1|$ no lugar de m, temos que $|B_2| \leq m - |B_1|$, implicando em $|B| \leq m$. De acordo com o Lema 4, também temos que $|B_2| \geq c \cdot (m - |B_1|)$ e, como estamos no Caso 2, então $d \leq \frac{1}{2}$ e $|P_z^b| \leq |S|$, dado que $P_z^b \subseteq S$. Note que a desigualdade $|T_z'| + |H_z^b| \leq \frac{1-d}{d}|P_z^b|$ implica que $|T_z'|\frac{d}{1-d} \leq |P_z^b|$. Portanto,

$$\begin{aligned} m - |B| &= m - |B_1| - |B_2| \\ &\leq (m - |B_1|) - c \cdot (m - |B_1|) = (m - |B_1|)(1 - c) \\ &\leq |T_z'|(1 - c) = |T_z'|\frac{d}{1-d} \leq |P_z^b| \leq |S|, \end{aligned}$$

o que implica que $m \leq |B| + |S|$ e, portanto, $|B| \leq m \leq |B| + |S|$.

Agora mostraremos que $e_T(B, W, S) \leq \frac{2 \cdot \Delta(T)}{\text{diam}^*(T)}$. Note que o número de arestas na partição (B, W, S) é, no máximo, $(\Delta(T) - 1) + 1 + 1 + e_{T_z'}(B_2, W_2) + (\Delta(T) - 2)$. Essa soma representa a quantidade total máxima de arestas no corte, considerando as que conectam z^b a W, a que liga z_l^b a B_1, a que conecta f e z, as presentes no corte (B_2, W_2) e, por fim, as arestas que ligam z a T_z' (talvez existam

arestas ligando B_2 a z). Logo, usando a desigualdade (10), temos que

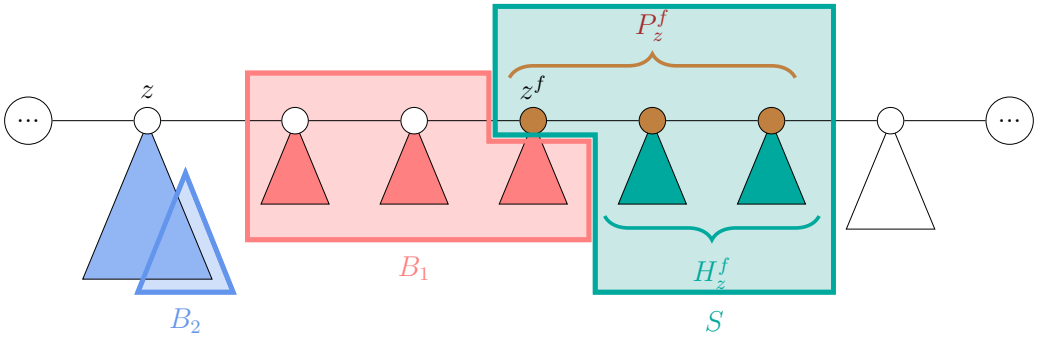
$$\begin{aligned} e_T(B,W,S) &= (\Delta(T)-1)+1+1+e_{T'_z}(B_2,W_2)+(\Delta(T)-2) \\ &= 2\cdot\Delta(T)+e_{T'_z}(B_2,W_2)-1 \\ &\leq 2\cdot\Delta(T)+\Big(\frac{2}{d}-3\Big)\Delta(T)-1=\Big(\frac{2}{d}-1\Big)\Delta(T)-1 \\ &< \frac{2\cdot\Delta(T)}{d}. \end{aligned}$$

Portanto, se a condição do Caso (a) for satisfeita, então existe uma partição (B,W,S) que satisfaz as propriedades do Teorema 1.

Caso (b): Existe um vértice $z \in F_{\text{esp}}$ tal que $|T'_z|+|H^f_z| \leq \frac{1-d}{d}|P^f_z|$. Temos que $|H^f_z| \leq \Big(\frac{1}{d}-1\Big)|P^f_z|-|T'_z|$. Usando a segunda desigualdade em (4), temos que $|H^f_z| \leq \Big(\frac{1}{2d}-1\Big)|P^f_z|$, o que implica que $|H^f_z|+|P^f_z| \leq \frac{1}{2d}|P^f_z|$.

Tomaremos $S = H^f_z \cup P^f_z$ e $B_1 = \{v \in V(T) : z < v < z^f\}$. Da mesma forma que no caso anterior, temos que $2d \leq \text{diam}^*(T[S])$.

Obtemos o corte (B_2,W_2) do Lema 4 aplicado a T'_z com $m-|B_1|$ e $2-\frac{1}{1-d}$ no lugar de m e c, respectivamente. Temos também que $B = B_1 \cup B_2$ e $W = V(T)\setminus(B\cup S)$. Dessa forma, a disposição dos vértices está ilustrada na figura a seguir.



Assim como foi mostrado no Caso (a), temos que $e_T(B, W, S) < \frac{2 \cdot \Delta(T)}{d}$, satisfazendo assim as propriedades do Teorema 1 em todos os casos apresentados anteriormente.

□

6.1 Algoritmo da dobra do diâmetro

O algoritmo aqui apresentado ilustra a função SEPARA_CONJUNTO(T, S), que remove de T as arestas que estão no corte $(S, V(T) \setminus S)$, separando S do restante da árvore T. Caso $T[S]$ seja conexo, v_i e v_j são os vértices em S de menor e maior rótulos, respectivamente. Caso contrário, v_i é o vértice de menor rótulo da componente de $T[S]$ que contém o final do caminho máximo P, e v_j é o vértice de maior rótulo da componente de $T[S]$ que contém o início de P. Sabemos que todos os vértices de S possuem rótulos consecutivos, considerando que os vértices estão dispostos de forma circular. Sabemos também que v_i e v_j possuem ao todo grau(v_i) arestas que os conectam a $V(T) \setminus S$. (Veja a figura anterior.) Portanto, o que essa função faz é remover de T as arestas que conectam v_i e v_j a $V(T) \setminus S$ e, caso $T[S]$ não seja conexo, inserir uma aresta em T que liga os vértices de menor e maior rótulos em T. Essa função devolverá um grafo T^*, que é a árvore T modificada, onde $T^*[S]$ será uma componente isolada do restante de T^*. Essa função consome tempo $O(\Delta(T))$.

Ao longo do Algoritmo 4, usaremos os conjuntos H_b^z, H_f^z, P_b^z, P_f^z e T'_z para diferentes vértices z. Para calcular cada um destes conjuntos, basta fazer operações de soma ou subtração nos rótulos para cada z. Como os conjuntos $P_z^b \cup H_z^b \cup T'_{z^b}$ são dois a dois disjuntos e os conjuntos $P_z^f \cup H_z^f \cup T'_{z^f}$ também são dois a dois disjuntos, então cada vértice da árvore será analisado no máximo duas vezes. Com isso, sabemos que a operação que calcula os vértices de cada um desses conjuntos para todos os valores de z consome tempo $O(n)$.

Serão passados como parâmetros dois vetores, rot e ind, que armazenam o rótulo de um vértice na posição de seu índice e o índice de um vértice na posição de seu rótulo, respectivamente.

Segue o algoritmo que encontra um corte que satisfaz as propriedades do Teorema 1.

Algoritmo 4:

entrada: árvore T com n vértices, $m \in [n]$, caminho máximo P em T, vetor rot, vetor ind e raiz r da árvore T

saída : partição (B, W, S) que satisfaz as propriedades do Teorema 1, uma árvore T^* que corresponde a $T[S]$ e um vértice $s \in S$

for $i = 1 \to |V(P)|$ **do**
  if $\text{ind}(\text{rot}(i) + m) \in V(P)$ **then**
    // Caso 1
    $B \leftarrow \{\text{ind}(\text{rot}(i) + 1), \ldots, \text{ind}(\text{rot}(i) + m)\}$;
    $W \leftarrow V(T) \setminus B$;
    return $[(B, W, \emptyset), \emptyset, null]$;
  end
end
// Caso 2
$valor_B \leftarrow \infty$;
for $z' \in B_{\text{esp}}$ **do**
  if $\frac{|T'_{z'}|+|H^b_{z'}|}{|P^b_{z'}|} < valor_B$ **then**
    $valor_B \leftarrow \frac{|T'_{z'}|+|H^b_{z'}|}{|P^b_{z'}|}$;
    $z_B \leftarrow z'$;
  end
end

15 $valor_F \leftarrow \infty$;
16 **for** $z' \in F_{\text{esp}}$ **do**
17 | **if** $\frac{|T'_{z'}|+|H^f_{z'}|}{|P^f_{z'}|} < valor_F$ **then**
18 | | $valor_F \leftarrow \frac{|T'_{z'}|+|H^f_{z'}|}{|P^f_{z'}|}$;
19 | | $z_F \leftarrow z'$;
| **end**
end
20 **if** $valor_B \leq valor_F$ **then**
| // Caso 2(a)
21 | $z \leftarrow z_B$;
22 | $z^b_\ell \leftarrow$ último vértice de P^b_z;
23 | $B_1 \leftarrow \{\text{ind}(\text{rot}(z^b_l)+1), \ldots, \text{ind}(\text{rot}(z^b_l)+m)\}$;
24 | $(B_2, W_2) \leftarrow$ **Algoritmo3**$(T'_z,\ |T'_z|,\ m-|B_1|,\ 2-\frac{1}{1-\text{diam}^*(T)})$;
25 | $B \leftarrow B_1 \cup B_2$;
26 | $W \leftarrow V \setminus (B \cup S)$;
27 | $S \leftarrow P^b_z \cup H^b_z$;
28 | $T^* \leftarrow$ SEPARA_CONJUNTO$(T,\ S)$;
29 | **return** $[(B, W, S),\ T^*,\ z^b_\ell]$;
end
30 **else**
| // Caso 2(b)
31 | $z \leftarrow z_F$;
32 | $z^f \leftarrow$ primeiro vértice de P^f_z;
33 | $B_1 \leftarrow \{\text{ind}(\text{rot}(z)+1), \ldots, \text{ind}(\text{rot}(z^f)-1)\}$;
34 | $(B_2, W_2) \leftarrow$ **Algoritmo3**$(T'_z,\ |T'_z|,\ m-|B_1|,\ 2-\frac{1}{1-\text{diam}^*(T)})$;
35 | $B \leftarrow B_1 \cup B_2$;
36 | $W \leftarrow V \setminus (B \cup S)$;
37 | $S \leftarrow P^f_z \cup H^f_z$;
38 | $T^* \leftarrow$ SEPARA_CONJUNTO$(T,\ S)$;
39 | **return** $[(B, W, S),\ T^*,\ z^f]$;
end

Análise do algoritmo

Dado que o algoritmo recebe como entrada o caminho máximo e a rotulação dos vértices da árvore induzida por esse caminho, precisamos apenas verificar em qual dos casos a árvore T se encaixa e encontrar a partição (B, W, S).

Primeiramente, verificamos se T se encaixa no Caso 1. Para isso é necessário percorrer todos os vértices do caminho máximo e verificar se algum deles é mapeado em um outro vértice do caminho máximo. Isso leva tempo linear no número de vértices no caminho máximo, que é $O(n)$.

Se a árvore T se encaixa no Caso 1, percorremos os vértices que serão devolvidos em tempo $O(n)$. Caso contrário, precisamos encontrar os vértices pertencentes aos conjuntos H_b^z, H_f^z, P_b^z, P_f^z e T'_z para cada z, e isso é feito em tempo $O(n)$, como mostrado anteriormente.

Feito isso, calculamos os valores $\dfrac{|T'_z| + |H_z^b|}{|P_z^b|}$ e $\dfrac{|T'_z| + |H_z^f|}{|P_z^f|}$ para todos os vértices b-especiais e f-especiais, respectivamente. Isso leva tempo linear no número de vértices b-especiais e f-especiais, que é $O(n)$.

Depois disso, tanto no Caso 2(a) quanto no 2(b), realizamos uma chamada ao Algoritmo 3, que leva tempo $O\Big(\Big\lceil \dfrac{2c}{1-c} \Big\rceil n + 1\Big)$, onde o valor passado como c é $2 - \dfrac{1}{1-d}$. Então temos que

$$\Big\lceil \frac{2c}{1-c} \Big\rceil n + 1 = \Big(\Big\lceil \frac{2}{d} \Big\rceil - 4 \Big) n + 1 \leq \Big\lceil \frac{2}{d} \Big\rceil n.$$

Logo, o Algoritmo 3 leva tempo $O\Big(\dfrac{n}{d}\Big)$.

Temos também a chamada a SEPARA_CONJUNTO(T, S), que leva tempo $O(\Delta(T))$, como mostrado nesta seção.

Sendo assim, se a árvore se encaixa no Caso 1, o consumo de tempo do Algoritmo 4 é $O(n)$. Caso contrário, serão executadas três operações de custo $O(n)$ e uma de custo $O\Big(\dfrac{n}{d}\Big)$, como mostrado anteriormente.

Como $d \leq 1$, temos que $\dfrac{n}{d} \geq n$, o que implica que o consumo de tempo do Caso 2 é $O\Big(\dfrac{n}{d}\Big)$, que é o mesmo que $O\Big(\dfrac{n}{\mathrm{diam}^*(T)}\Big)$.

Portanto, é possível encontrar uma partição (B, W, S) que satisfaça as

condições do teorema em tempo $O\Big(\frac{n}{\text{diam}^*(T)}\Big)$.

7 Algoritmo FST para bissecção

Nesta seção, finalmente apresentamos o algoritmo FST. Ele utiliza o Algoritmo 4 repetidas vezes além de manter informações derivadas do caminho máximo e da rotulação inicial.

Proposição 1. *Para toda árvore T com $n > 2$ vértices e todo par de vértices v_1 e v_2 em T com $\text{grau}_T(v_1) = \text{grau}_T(v_2) = 1$, temos que $\Delta(T) = \Delta(T + e)$, onde $e = \{v_1, v_2\}$.*

Demonstração. Como $n > 2$, temos que $\Delta(T) \geq 2$. Sabemos que ao adicionar $e = \{v_1, v_2\}$ em T, o grau de v_1 e v_2 passará a ser 2, e os demais vértices de T irão manter os seus respectivos graus. Portanto, $\text{grau}_{T+e}(v_1) = \text{grau}_{T+e}(v_2) = 2$ e, para todo vértice v tal que $v \neq v_1$ e $v \neq v_2$, temos que $\text{grau}_{T+e}(v) = \text{grau}_T(v)$. Logo $\Delta(T + e) = \Delta(T)$. □

Segue o teorema principal.

Teorema 2 (Teorema do corte exato [10, Teorema 6])**.** *Para todo $i \in \mathbb{N}_*^+$, toda árvore T com n vértices e $\text{diam}^*(T) > \frac{1}{2^i}$, e todo $m \in [n]$, existe um corte (B, W) em T com $|B| = m$ e $e_T(B, W) \leq 4 \cdot 2^i \cdot \Delta(T)$.*

Demonstração. Utilizaremos indução em i para provar o Teorema 2.

Primeiramente, temos que $\text{diam}^*(T) \leq 1$, por definição. Sendo assim, $i \geq 1$, logo $i = 1$ será a base utilizada na indução.

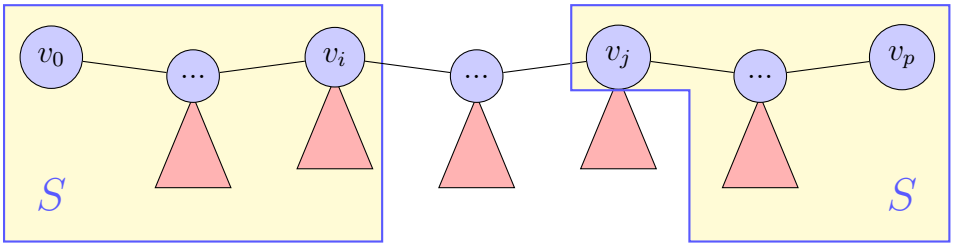
Base: i = 1. Como $\text{diam}^*(T) > \frac{1}{2}$, podemos utilizar o Lema 5 que garante a existência de um corte (B, W) com $|B| = m$ e $e_T(B, W) \leq 2$. Note que, como $\text{diam}^*(T) > \frac{1}{2}$, T possui mais que um único vértice, logo $\Delta(T) \geq 1$ e $e_T(B, W) \leq 4 \cdot 2^i \cdot \Delta(T)$. Portanto, a afirmação vale para $i = 1$.

Passo: i $\geq$ 2. Para $n \leq 2$, sabe-se que $|E(T)| = \Delta(T)$. Portanto, para $n \leq 2$, temos que $e_T(B, W) \leq |E(T)| = \Delta(T) \leq 2 \cdot 2^i \cdot \Delta(T)$. Se $n > 2$, podemos aplicar o Teorema 1, que garante a existência de uma partição (B', W', S') do tipo 1 ou 2.

Se existir uma partição (B', W', S') do tipo 1, teremos $|B'| = m$, $S' = \emptyset$ e $e_T(B', W') \leq 2 \leq 2 \cdot 2^i \cdot \Delta(T)$. Logo, o corte (B', W') satisfaz as propriedades do corte do teorema.

Caso contrário, a partição (B', W', S') será do tipo 2, ou seja, $|B'| = m$ ou $|B'| < m \leq |B'| + |S'|$, e $e_T(B', W', S') \leq \dfrac{2 \cdot \Delta(T)}{\text{diam}^*(T)} \leq 2 \cdot 2^i \cdot \Delta(T)$.

Se $|B'| = m$, então o corte $(B', V \setminus B')$ satisfaz as propriedades do corte do teorema. Senão, $|B'| < m \leq |B'| + |S'|$, que implica que $0 < m - |B'| \leq |S'|$, e $\text{diam}^*(T[S']) \geq 2 \cdot \text{diam}^*(T) > \dfrac{1}{2^{i-1}}$. Assim, se $T[S']$ for uma árvore, podemos aplicar a hipótese de indução a $T[S']$ usando $m - |B'|$ como m. Porém, é possível que $T[S']$ não seja conexo. De fato, examinando-se a prova do Teorema 1, $T[S']$ pode se dividir em duas componentes, uma no começo e outra no fim do caminho máximo utilizado. Isso pode ser visto na figura abaixo.



Como o Teorema 2 se aplica a árvores, precisamos tornar $T[S']$ conexo sem alterar o seu grau máximo nem o seu diâmetro relativo. Sabemos que se adicionarmos uma aresta e em T que liga o primeiro ao último vértice do caminho máximo P usado na prova do Teorema 1, então $T[S']$ será conexo. Diremos que $T^* = T + e$.

Observe que o Algoritmo 4 já faz esse acerto em $T[S']$ antes de devolvê-la. No Algoritmo 5, temos que retirar arestas da árvore para desconectar os vértices de S' do resto da árvore e acrescentar a aresta e, como descrito na prova do teorema. Poderíamos desconectar os vértices de S' do restante

verificando todas as arestas de T que ligam dois vértices de S' e fazer uma cópia de $T[S']$, mas podemos fazer isso no Algoritmo 4 de forma mais eficiente. No Algoritmo 4, analisamos os vértices de S' que se ligam a vértices de $V \setminus S'$, fazendo uma marcação para indicar que as arestas que ligam S' e $V \setminus S'$ estão inativas, tornando assim S' uma componente conexa isolada. Obter a árvore T^* acrescentando a aresta e em T, como descrito no Teorema 2, também pode ser feito de forma mais simples no Algoritmo 4, como no procedimento acima. Desta forma, o Algoritmo 4 retornará a árvore $T^*[S']$ separada do resto de T.

Note também que em ambas as componentes de $T[S']$, temos que o caminho máximo em cada uma possui em um dos seus extremos o vértice v_0 ou o vértice v_p. Logo, $T^*[S']$ será conexo e teremos um caminho em $T^*[S']$ com comprimento adequado. Além disso, de acordo com a Proposição 1, temos que $\Delta(T) = \Delta(T^*)$, já que a aresta e conecta os dois vértices extremos do caminho máximo e estes possuem grau um. Podemos, então, aplicar a indução à árvore $T^*[S']$ sabendo que não há aumento do grau máximo nem do diâmetro relativo. Ou seja, existe um corte (B'', W'') em $T^*[S']$ tal que $|B''| = m - |B'|$ e $e_{T[S]}(B'', W'') \leq 4 \cdot 2^{i-1} \cdot \Delta(T^*[S']) \leq 4 \cdot 2^{i-1} \cdot \Delta(T^*)$.

Dessa forma, o corte (B, W) em T com $B = B' \cup B''$ e $W = V(T) \setminus B$ é tal que $|B| = |B'| + m - |B'| = m$ e

$$\begin{aligned} e_T(B, W) &\leq e_T(B', W', S') + e_{T[S]}(B'', W'') \\ &\leq 2 \cdot 2^i \cdot \Delta(T) + 4 \cdot 2^{i-1} \cdot \Delta(T^*) \\ &= 2 \cdot 2^i \cdot \Delta(T) + 4 \cdot 2^{i-1} \cdot \Delta(T) \\ &= 4 \cdot 2^i \cdot \Delta(T), \end{aligned}$$

completando a prova por indução do teorema. □

Corolário 1 ([10, Corolário 7])**.** *Para toda árvore T com n vértices e todo $m \in [n]$, existe um corte (B, W) em T com $|B| = m$ e $e_T(B, W) \leq \dfrac{8}{\operatorname{diam}^*(T)} \Delta(T)$.*

Demonstração. Para toda árvore T, como $0 < \text{diam}^*(T) \leq 1$, existe um j tal que $\frac{1}{2^j} < \text{diam}^*(T) \leq \frac{1}{2^{j-1}}$, o que implica que $2^j \leq \frac{2}{\text{diam}^*(T)}$.

Assim, o Teorema 2 garante a existência de um corte (B, W) em T com $|B| = m$ e $e_T(B, W) \leq 4 \cdot 2^j \cdot \Delta(T) \leq \frac{8}{\text{diam}^*(T)}\Delta(T)$, e esse corte satisfaz as propriedades do Corolário 1. □

7.1 Algoritmo FST

O algoritmo FST faz uso das seguintes rotinas.

A função CAMINHO_MÁXIMO(T) recebe uma árvore T com n vértices e devolve um vetor contendo os vértices de um caminho máximo em T, de acordo com a ordem apresentada no caminho. Isso pode ser feito em tempo $O(n)$, como descrito na Seção 3.

A função ROTULA(T, c) recebe uma árvore T com n vértices e um vetor c contendo os vértices de um caminho em T e devolve dois vetores: o primeiro contém a rotulação dos vértices de T descrita na Seção 5, e o segundo, o inverso da rotulação. Esta função consome tempo $O(n)$, como mostrado na Seção 5.

Em vez de calcularmos novamente o caminho máximo da nova árvore, podemos reutilizar o caminho calculado anteriormente, aproveitando somente a parte do caminho que está contida em $T^*[S']$. Se usarmos essa parte no lugar de um caminho máximo, manteremos a propriedade da dobra do diâmetro como pode ser visto na segunda desigualdade de (9) na Seção 6. Dessa forma, a rotulação continua basicamente a mesma, bastando apenas fazer um ajuste nos vértices para a esquerda, para que os rótulos comecem de 1.

Para reutilizar o caminho da árvore antiga, utilizamos a função RECALCULA_CAMINHO(T, S, c), que recebe como parâmetros uma árvore T, um conjunto de vértices $S \subseteq V(T)$ e um vetor c, que contém os vértices de um caminho na árvore T ordenados de acordo com a sua posição neste

caminho. Essa função devolve um vetor que contém os vértices de $S \cap c$, na mesma ordem que aparecem em c. Isso pode ser feito em tempo $O(|c|)$ se o conjunto S é dado por um vetor binário.

Segue o algoritmo FST que encontra um corte que satisfaz as propriedades do Teorema 2.

Algoritmo 5:

entrada: árvore $T = (V, E)$ com n vértices e $m \in [n]$
saída : corte (B, W) tal que $|B| = m$ e $e_T(B, W) \leq \frac{8}{\text{diam}^*(T)}\Delta(T)$

```
1  B ← ∅;
2  T' ← T;
3  r ← vértice arbitrário de T;
4  P ← CAMINHO_MÁXIMO(T);
5  [rot, ind] ← ROTULA(T, P);
6  while |B| < m do
7  |   [(B', W, S), T', r] ← Algoritmo4(T', m − |B|, P, rot, ind, r);
8  |   B ← B ∪ B';
9  |   P ← RECALCULA_CAMINHO(T', S, P);
10 |   [rot, ind] ← ROTULA(T', P)
11 end
12 return (B, V \ B);
```

Análise do Algoritmo

Dado que, inicialmente, $B = \emptyset$, o laço das linhas 6-11 sempre será executado. Seja $T[S]_1$ o grafo $T[S]$ passado como parâmetro para o Algoritmo 4. De acordo com o Teorema 1, temos que em cada execução da linha 7, o corte (B', W, S) tem as propriedades de um dos seguintes itens:

1. $|B'| = m - |B|$, $S = \emptyset$ e $e_T(B', W) \leq 2$ ou
2. $|B'| \leq m - |B| \leq |B'| + |S|$, com $0 < |S| \leq \frac{n}{2}$, $e_T(B, W, S) \leq \frac{2 \cdot \Delta(T[S]_1)}{\text{diam}^*(T[S]_1)}$, e $\text{diam}^*(T[S]) \geq 2\ \text{diam}^*(T[S]_1)$.

Se em alguma iteração do laço das linhas 6-11, o corte (B', W, S) retornado pelo Algoritmo 4 for tal que $|B'| = m - |B|$, teremos que $|B| = m$ na linha 8 e, desta forma, na linha 12 será retornado um corte (B, W) com $|B| = m$ e $e_T(B, W) \leq \dfrac{2 \cdot \Delta(T)}{\text{diam}^*(T)} \leq \dfrac{8}{\text{diam}^*(T)}\Delta(T)$.

Caso o corte (B', W, S) obtido na linha 7 seja tal que $|B'| < m - |B|$, sabe-se que este segue as propriedades do item 2 e portanto, $\text{diam}^*(T[S]) \geq 2\ \text{diam}^*(T[S]_1)$. Como o diâmetro relativo de $T[S]$ irá, no mínimo, dobrar a cada chamada ao Algoritmo 4, em algum momento $\text{diam}^*(T) > \dfrac{1}{2}$, fazendo com que o Algoritmo 4 retorne um corte que condiz com o item 1, como visto anteriormente nesta seção. Portanto, teremos que $|B| = m$ na linha 8. Assim, o algoritmo retorna um corte (B, W) tal que $|B| = m$ e $e_T(B, W) \leq \dfrac{8}{\text{diam}^*(T)}\Delta(T)$, como mostrado no Corolário 1.

Sabe-se que em nenhum momento $|B| > m$ irá ocorrer, dado que a chamada ao algoritmo 4 da linha 7 sempre retorna um corte (B', W, S) de forma que $|B'| \leq m - |B|$.

Agora vamos analisar o consumo de tempo do algoritmo. Considere que ℓ seja o número de vezes que o laço das linhas 6-11 foi executado, com $\ell = 0$ inicialmente. Temos que T_ℓ e n_ℓ representam a árvore da iteração ℓ e o número de vértices dessa árvore, respectivamente. Sabemos que o Algoritmo 5 executa o Algoritmo 4 que consome tempo $O\Big(\dfrac{n_\ell}{\text{diam}^*(T_\ell)}\Big)$, a função RECALCULA_CAMINHO que consome tempo $O(n_\ell)$, e a função ROTULA que consome tempo $O(n_\ell)$. Sendo i um inteiro tal que $\dfrac{1}{2^i} < \text{diam}^*(T) \leq \dfrac{1}{2^{i-1}}$, temos que cada uma das operações citadas anteriormente será executada no máximo i vezes, com $\ell = 0, 1, \cdots, \ell - 1$.

Sabe-se que o número de vértices de T_ℓ é no mínimo o dobro do número de vértices de $T_{\ell+1}$. Com isso, provaremos por indução que $n_\ell \leq \dfrac{n}{2^\ell}$ para $\ell \geq 0$.

Demonstração. **Base: $\boldsymbol{\ell = 0}$.** Dado que inicialmente $T'[S] = T$, temos que $n_0 = n$, o que implica que $n_0 = \dfrac{n}{2^0}$.

Passo: $\ell > 0$. Sabendo que $2 \cdot n_\ell \leq n_{\ell-1}$, temos que

$$\begin{aligned} 2 \cdot n_\ell &\leq n_{\ell-1} \leq \frac{n}{2^{\ell-1}} \\ n_\ell &\leq \frac{n}{2^\ell}, \end{aligned}$$

completando a prova de indução. □

Assim, para todo $\ell \in [i]$, temos que $n_\ell \leq \frac{n}{2^\ell}$. Portanto o tempo de execução do Algoritmo 5 é

$$\begin{aligned} \sum_{\ell=1}^{i} \Big(O\Big(\frac{n_\ell}{\mathrm{diam}^*(T_\ell)}\Big) + O(n_\ell) + O(n_\ell) \Big) &= \sum_{\ell=1}^{i} O\Big(\frac{n_\ell}{\mathrm{diam}^*(T_\ell)}\Big) \\ &= \sum_{\ell=1}^{i} O\Big(\frac{n_\ell}{\mathrm{diam}^*(T)}\Big) \\ &= \sum_{\ell=1}^{i} O\Big(\frac{1}{2^\ell} \cdot \frac{n}{\mathrm{diam}^*(T)}\Big) \\ &= O\Big(\frac{n}{\mathrm{diam}^*(T)}\Big). \end{aligned}$$

Logo, o algoritmo leva tempo $O\Big(\frac{n}{\mathrm{diam}^*(T)}\Big)$.

8 Complexidade do problema da bissecção

Nesta seção apresentaremos a demonstração de que o problema da bissecção mínima é NP-difícil. Esse resultado deve-se a Garey, Johnson e Stockmeyer [5], e foi obtido através de duas reduções. Os autores demonstraram primeiro que o problema MAXCUT, definido abaixo, é NP-difícil, a partir de uma redução do fundamental MAX2SAT, também definido abaixo, e em seguida demonstraram que o problema da bissecção mínima é NP-difícil a partir de uma redução de MAXCUT.

Optamos por apresentar essas duas reduções neste trabalho. Começamos definindo os problemas usados.

Problema 1 (MAX2SAT-D(C, k) – Versão de decisão de satisfatibilidade máxima com no máximo dois literais por cláusula)**.** *Dados um conjunto C com p cláusulas distintas representando uma fórmula booleana na forma normal disjuntiva, com no máximo dois literais cada, e um inteiro $k \leq p$, decidir se existe uma atribuição de valores para as variáveis de C de forma que k ou mais cláusulas de C sejam verdadeiras.*

Problema 2 (MAXCUT-D(G, W) – Versão de decisão do corte máximo)**.** *Dados um grafo G e um inteiro positivo W, decidir se existe um corte (T, F) em G tal que $e_G(T, F) \geq W$.*

Karp [8] demonstrou que o problema MAX3SAT é NP-difícil, e isso foi utilizado por Garey, Johnson e Stockmeyer [5] para provar que o problema MAX2SAT é NP-difícil. Usaremos esse fato na demonstração do seguinte.

8.1 Redução do MAX2SAT para o MAXCUT

Teorema 3. MAXCUT *é NP-difícil.*

Demonstração. Vamos apresentar uma redução polinomial do MAX2SAT-D para o MAXCUT-D.

Dada uma instância (C, k) do MAX2SAT-D, precisamos construir uma instância (G, W) do problema MAXCUT-D, ou seja, um grafo $G = (V, E)$ e um inteiro positivo W.

Seja p o número de cláusulas em C e n o número de variáveis em C. Denote por $x_1, x_2, \ldots, x_n$ as variáveis de C. Lembre-se que $\overline{x}$ é a negação da variável x e que um literal de C é uma variável de C ou a negação de uma variável de C.

O conjunto de vértices de G é definido da seguinte maneira:

$$\begin{aligned} V = &\{x_i : 1 \leq i \leq n\} \cup \\ &\{\overline{x_i} : 1 \leq i \leq n\} \cup \\ &\{T_j : 0 \leq j \leq 3p\} \cup \\ &\{F_j : 0 \leq j \leq 3p\} \cup \\ &\{t_{ij} : 1 \leq i \leq n,\ 0 \leq j \leq 3p\} \cup \\ &\{f_{ij} : 1 \leq i \leq n,\ 0 \leq j \leq 3p\}. \end{aligned}$$

O conjunto de arestas de G é dado em partes. Primeiro, considere

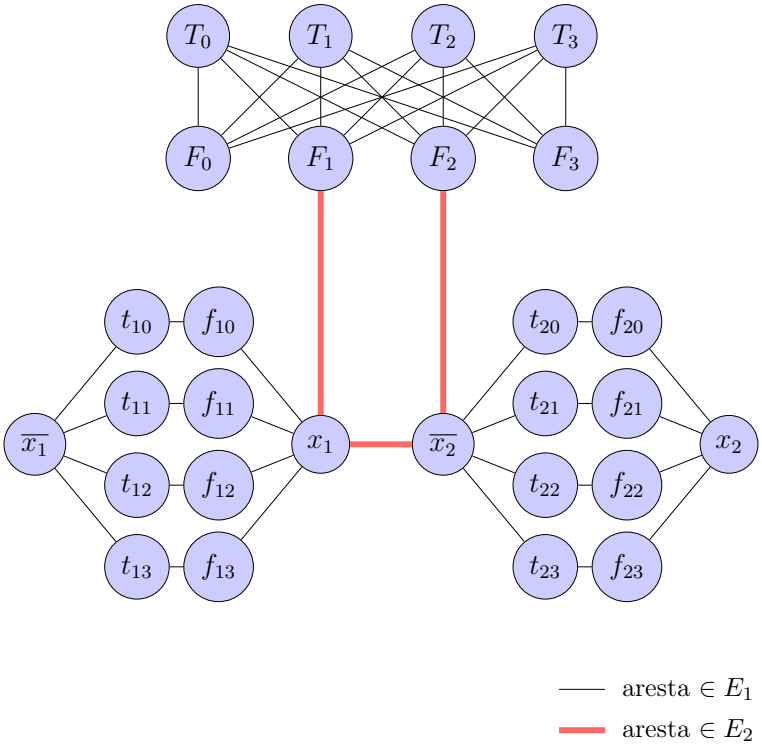
$$\begin{aligned} E_1 = &\{\{T_i, F_j\} : 0 \leq i \leq 3p,\ 0 \leq j \leq 3p\} \cup \\ &\{\{t_{ij}, f_{ij}\} : 1 \leq i \leq n,\ 0 \leq j \leq 3p\} \cup \\ &\{\{x_i, f_{ij}\} : 1 \leq i \leq n,\ 0 \leq j \leq 3p\} \cup \\ &\{\{\overline{x_i}, t_{ij}\} : 1 \leq i \leq n,\ 0 \leq j \leq 3p\}. \end{aligned}$$

Agora, sejam $C_1, C_2, \ldots, C_p$ as cláusulas de C e sejam a_i e b_i os literais de C_i para $i = 1, 2, \ldots, p$. Considere então

$$\begin{aligned} E_2 = &\{\{a_i, b_i\} : 1 \leq i \leq p,\ a_i \neq b_i\} \cup \\ &\{\{a_i, F_{2i-1}\} : 1 \leq i \leq p\} \cup \\ &\{\{b_i, F_{2i}\} : 1 \leq i \leq p\}. \end{aligned}$$

Dessa forma, temos o grafo $G = (V, E)$, com $E = E_1 \cup E_2$, e o inteiro positivo $W = |E_1| + 2k$. Note que o grafo é simples pois não há cláusulas repetidas em C.

Segue abaixo um exemplo de grafo G obtido da cláusula $(x_1 \vee \overline{x_2})$.



Claramente G e W podem ser obtidos de (C, k) em tempo polinomial em p e n. Resta mostrar que existe uma atribuição de valores para as variáveis de C que satisfaz pelo menos k cláusulas de C se e somente se existe um corte (T, F) em G tal que $e_G(T, F) \geq W$.

Agora vamos supor que temos uma atribuição de valores para as variáveis de C que tornam k ou mais cláusulas de C verdadeiras.

Note que existem no máximo $2p$ arestas do tipo $\{x_i, F_j\}$ ou $\{\overline{x_i}, F_j\}$. Dado que $2p < 3p + 1$ e as funções $f(i) = 2i - 1$ e $g(i) = 2i$ têm imagens disjuntas e são injetoras, temos que cada F_j se liga a apenas um vértice do tipo x_i ou $\overline{x_i}$.

Definiremos agora os conjuntos T e F de um corte (T, F) de G. Para todo $i \in [0, 3p]$, $T_i \in T$ e $F_i \in F$ e, para todo $j \in [1, n]$ e $k \in [0, 3p]$, x_j per-

tence ao mesmo conjunto que t_{jk} e ambos não pertencem ao mesmo conjunto que $\overline{x_j}$ e que f_{jk}. Assim sendo, todas as arestas de E_1 estão no corte (T, F). Ademais, se a variável x_i é verdadeira, então o vértice x_i está em T, caso contrário, x_i está em F. Isso não contraria as escolhas feitas anteriormente, portanto obtemos um corte (T, F).

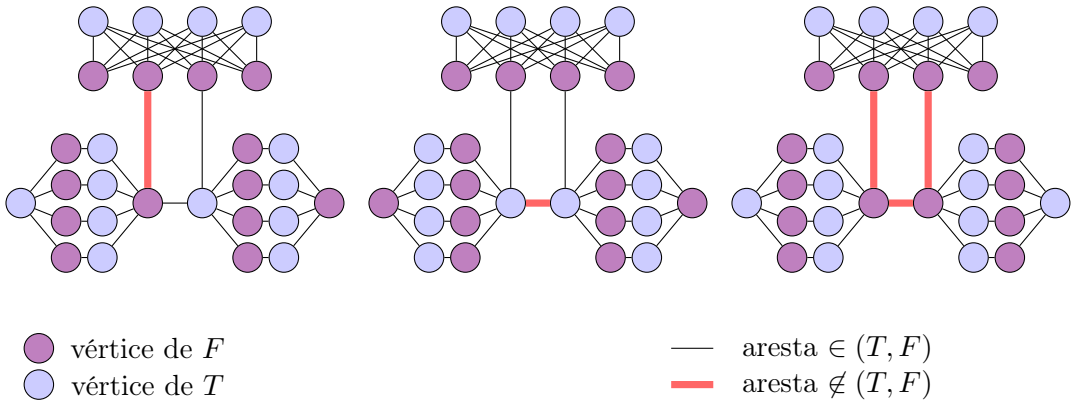
Em uma cláusula $C_i = (a_i, b_i)$, ou apenas um dos literais é verdadeiro, ou ambos são, ou nenhum deles é.

1. Se apenas um dos literais é verdadeiro, das três arestas formadas pela cláusula C_i, $\{a_i, b_i\}$, $\{a_i, F_{2i-1}\}$ e $\{b_i, F_{2i}\}$, duas delas estão no corte (T, F) e uma está fora.

2. Se os dois literais são verdadeiros, também teremos duas arestas no corte (T, F) e uma fora.

3. Já no caso de nenhum literal ser verdadeiro, nenhuma dessas arestas da cláusula estará no corte (T, F).

Podemos ver isso de forma mais clara na imagem a seguir.



Note que temos $2k$ ou mais arestas de E_2 no corte (T, F), pois, para cada uma das k cláusulas verdadeiras, teremos duas arestas de E_2 diferentes no corte, como mostrado anteriormente. Portanto, no mínimo $|E_1| + 2k = W$ arestas de G estão no corte (T, F).

Agora vamos supor que (T, F) é um corte em G, com $e_G(T, F) \geq W$. Temos que provar que existem no mínimo k cláusulas verdadeiras em C.

Note que $|E_2| = 3p$, pois o conjunto E_2 é composto por três arestas de cada uma das cláusulas de C. Se os vértices do tipo T_j e os do tipo F_j não ficarem separados no corte (T, F), no mínimo $3p + 1$ arestas de E_1 não estarão no corte (T, F) e, consequentemente, $e_G(T, F) \leq |E_1| - (3p + 1) + |E_2| < |E_1| \leq W$, uma contradição. Portanto, os vértices do tipo T_j e F_j estão em conjuntos diferentes no corte (T, F) e todas as arestas do tipo $\{T_i, F_j\}$ estão no corte (T, F).

De maneira similar, podemos ver que, para toda variável x_i, os vértices x_i e $\overline{x_i}$ de V estão em conjuntos diferentes no corte (T, F). Existem $3p + 1$ caminhos diferentes entre x_i e $\overline{x_i}$, e cada um deles contém quatro vértices (x_i, f_{ij}, t_{ij} e $\overline{x_i}$). Se x_i pertence ao mesmo conjunto que $\overline{x_i}$, pelo menos uma aresta de cada caminho não estará no corte (T, F), ou seja, no mínimo $3p + 1$ arestas de E_1 não estarão no corte (T, F), o que implica que $e_G(T, F) < W$, como visto anteriormente, uma contradição.

Considere a seguinte atribuição de valores para as variáveis de C. Se o vértice $x_i \in T$, a variável x_i tem valor verdadeiro. Caso contrário, $\overline{x_i} \in T$ e a variável x_i tem valor falso.

Como todos os vértices do tipo F_i estão no mesmo conjunto do corte (T, F), e sabendo que cada cláusula gera um caminho do tipo F_m, x_n, x_o, F_p, então para cada cláusula, zero (cláusula falsa) ou duas (cláusula verdadeira) arestas estão no corte (T, F). Assim, como $e_G(T, F) \geq W = |E_1| + 2k$, no mínimo k cláusulas serão verdadeiras para essa atribuição.

Esses valores das variáveis de C podem ser obtidos de G em tempo polinomial em n. □

8.2 Redução do MAXCUT para a Bissecção

Problema 3 (Bissecção(G, W) – Versão de decisão da bissecção mínima)**.** *Dados um grafo G e um inteiro positivo W, decidir se existe um corte (R, S)*

em G tal que $|R| = |S|$ e $e_G(R, S) \leq W$.

Teorema 4. *O problema da bissecção mínima é NP-difícil.*

Demonstração. Vamos apresentar uma redução polinomial do MAXCUT-D para o problema da Bissecção.

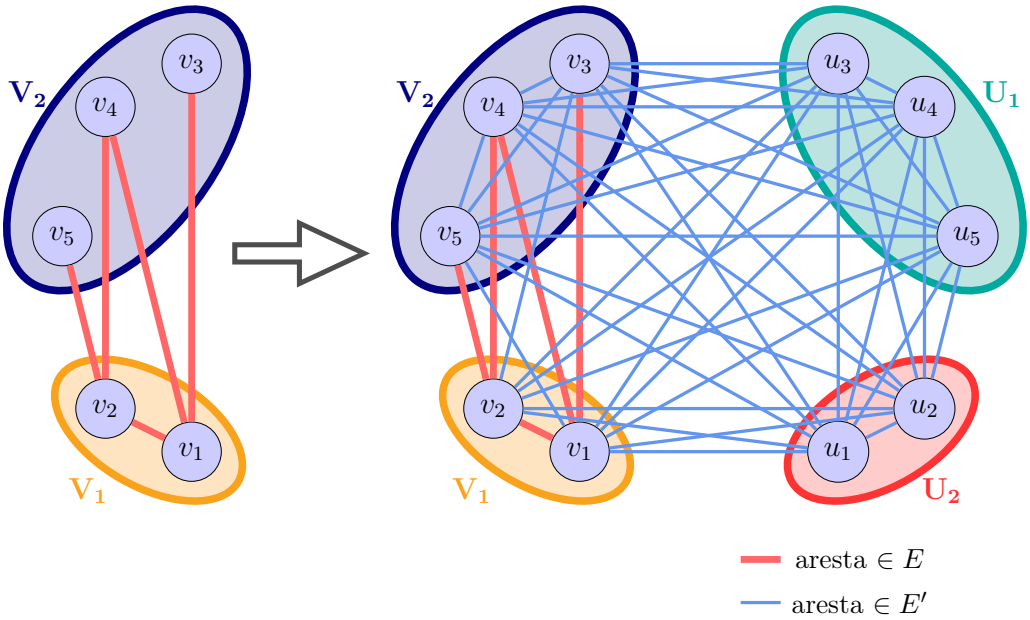
Dada uma instância (G, W) do problema MAXCUT-D, com $G = (V, E)$, precisamos construir uma instância (G', W') do problema da Bissecção, com $G' = (V', E')$.

Seja $n = |V|$ e $U = \{u_1, u_2, \ldots, u_n\}$ com $V \cap U = \emptyset$. O conjunto de vértices e arestas de G' é dado da seguinte maneira:

$$V' = V \cup U \text{ e}$$
$$E' = \{\{x, y\} : x, y \in V' \text{ e } \{x, y\} \notin E\},$$

enquanto que $W' = n^2 - W$.

Segue um exemplo de grafo G' à direita (desconsiderando as arestas de E), para G à esquerda.



Note que G' e W' podem ser obtidos de G e W em tempo polinomial em n.

Agora vamos supor que existe um corte (V_1, V_2) em G, onde $e_G(V_1, V_2) \geq W$.

Denotemos por $U_1 \subseteq U$ e $U_2 \subseteq U$ dois conjuntos de vértices arbitrários tais que $|U_1| = |V_2|$, $|U_2| = |V_1|$ e $U_1 \cap U_2 = \emptyset$, como o ilustrado na figura acima. Seja (R, S) o corte de G' com $R = V_1 \cup U_1$ e $S = V_2 \cup U_2$. Note que o corte (R, S) é uma bissecção, dado que $|R| = |V_1| + |U_1| = |U_2| + |V_2| = |S|$.

Se G' fosse um grafo completo, $e_{G'}(R, S) = n^2$, mas G' não é completo, dado que as arestas de G não estão presentes em G'. Portanto, $e_{G'}(R, S) = n^2 - e_G(V_1, V_2) \leq n^2 - W = W'$.

Agora vamos supor que exista uma bissecção (R, S) tal que $e_G(R, S) \leq W'$. Temos que provar que existe um corte (V_1, V_2) em G tal que $e_G(V_1, V_2) \geq W$.

Sejam $V_1 = R \cap V$, $V_2 = S \cap V$ e $k = e_G(V_1, V_2)$. Se o grafo G' fosse completo, $e_{G'}(R, S) = n^2$, pois $|R| = |S| = n$. Como as arestas de E não estão em E', então $e_{G'}(R, S) = n^2 - k$.

Dado que $e_{G'}(R, S) \leq W'$ e que $W' = n^2 - W$, temos que $n^2 - k \leq n^2 - W$, o que implica que $W \leq k$. Portanto $e_G(V_1, V_2) = k \geq W$. □

9 Algoritmo de Jansen et al.

O algoritmo de Jansen et al. [6] baseia-se em programação dinâmica e sempre encontra uma solução ótima para o problema da bissecção mínima quando o grafo dado é planar. Seu consumo de tempo é $O(n^3)$, onde n é o número de vértices do grafo dado.

Como o foco deste trabalho é o problema da bissecção mínima em árvores, e dado que toda árvore é um grafo planar, nesta seção, apresentaremos a recorrência do algoritmo de Jansen et al. especializada para árvores.

Seja T uma árvore com n vértices e com uma raiz $r \in V(T)$. Dado um vértice u, seja T_u a sub-árvore de T que contém u e todos os seus descendentes. Para todo $v \in V(T)$ e todo $m \in [n]$, a função $b(v, m)$ representa o valor de $e_{T_v}(B, W)$, onde $m = |B|$ e (B, W) é um corte de largura mínima em T_v com $v \in B$. Se $m > |V(T_v)|$, então não há corte (B, W) em T_v com $|B| = m$ e estabelecemos que $b(v, m) = \infty$. O mesmo vale para a função $w(v, m)$, mudando somente o fato de que, neste caso, exigimos que $v \in W$. Ou seja, $w(v, w)$ representa o valor de $e_{T_v}(B, W)$, onde $m = |B|$ e (B, W) é um corte de largura mínima em T_v com $v \in W$. Se $m \geq |V(T_v)|$, então $w(v, m) = \infty$.

Sendo assim, para todo vértice v, é válido que:

- Se v é uma folha de T, então

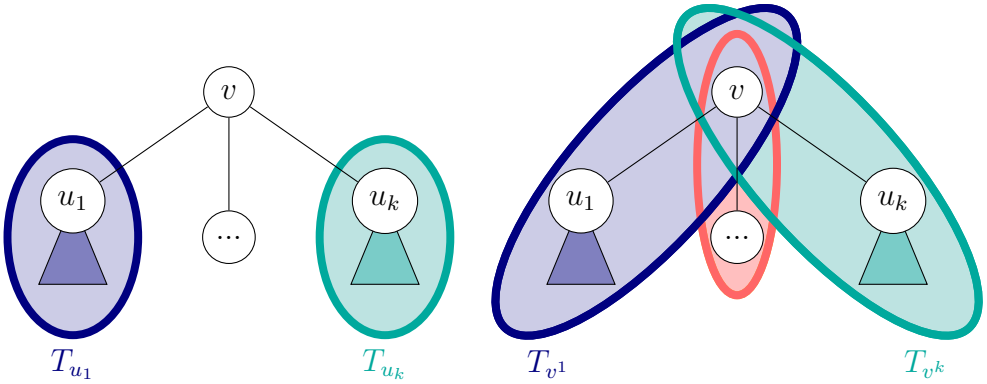
  $$b(v, m) = \begin{cases} 0, & \text{se } m = 1 \\ \infty, & \text{se } m \neq 1 \end{cases}$$

  $$w(v, m) = \begin{cases} 0, & \text{se } m = 0 \\ \infty, & \text{se } m \neq 0, \end{cases}$$

  dado que, como v é uma folha, os únicos cortes possíveis em T_v são $(B, W) = (\{v\}, \emptyset)$ e $(B, W) = (\emptyset, \{v\})$.

  A função $b(v, m)$ requer que $v \in B$, então, só temos solução quando $m = 1$. Já a função $w(v, m)$ requer que $v \in W$, logo o único valor viável para m é 0. Ambas as soluções não possuem arestas de T_v no corte, dado que a árvore T_v possui um único vértice.

- Se v não é uma folha de T, chame de $u_1, u_2, \ldots, u_k$ os filhos de v. Seja T_{v^i} a árvore T_{u_i} acrescida do vértice v e da aresta $\{v, u_i\}$. como mostrado na figura abaixo.



Para facilitar o entendimento das funções $b(v, m)$ e $w(v, m)$, vamos quebrá-las usando mais dois pares de fórmulas.

O primeiro par é $\tilde{b}_i(v, m)$ e $\tilde{w}_i(v, m)$. O valor dessas funções é a largura de um corte mínimo (B, W) em T_{v^i} tal que $m = |B|$, sendo que em $\tilde{b}_i(v, m)$ exigimos que $v \in B$ e em $\tilde{w}_i(v, m)$ exigimos que $v \in W$.

Essas funções satisfazem o seguinte:

$$\tilde{b}_i(v, m) = \min\{b(u_i, m-1),\ w(u_i, m-1) + 1\} \text{ e}$$
$$\tilde{w}_i(v, m) = \min\{b(u_i, m) + 1,\ w(u_i, m)\}.$$

A função $\tilde{b}_i(v, m)$ requer que $v \in B$, logo, para obter um conjunto B com m vértices é necessário que o corte (B', W') em T_{u_i} seja tal que $|B'| = m - 1$. Desta forma, tomamos $B = B' \cup \{v\}$ e $W = W'$, implicando em $|B| = m$. Caso $u_i \in W$, a aresta $\{v, u_i\}$ estará no corte (B, W) (dado que $v \in B$), e por isso acrescentamos 1 ao valor de $w(u_i, m-1)$. Caso contrário, v e u_i pertencerão ao conjunto B e nenhuma nova aresta será adiciona ao corte.

Já a função $\tilde{w}_i(v, m)$ requer que $v \in W$. Para que tenhamos um conjunto B com m vértices, precisamos encontrar um corte (B', W') em T_{u_i} tal que $|B'| = m$. Assim, tomamos $B = B'$ e $W = W' \cup \{v\}$. Se $u_i \in B$, a aresta $\{v, u_i\}$ estará no corte (B, W) e, caso contrário, os dois vértices estarão em W e nenhuma aresta será adicionada.

O segundo par de funções é $\tilde{\tilde{b}}_i(v,m)$ e $\tilde{\tilde{w}}_i(v,m)$. O valor dessas funções é a largura de um corte mínimo (B,W) na árvore $T_{v^1} \cup T_{v^2} \cup \cdots \cup T_{v^i}$ com $v \in B$ na primeira função e $v \in W$ na segunda. A condição inicial de ambas é quando $i = 1$, e nesse caso temos que

$$\tilde{\tilde{b}}_1(v,m) = \tilde{b}_1(v,m) \quad \text{e} \quad \tilde{\tilde{w}}_1(v,m) = \tilde{w}_1(v,m),$$

pois a árvore em questão é exatamente a árvore T_{v^1}.

Seja (B_i, W_i) um corte de largura mínima em T_{v^i} com $|B_i| = m_i$. Para calcular as funções $b(v,m)$ e $w(v,m)$, poderíamos verificar todas as combinações possíveis de valores $m_1, m_2, \ldots, m_i$ tais que $m_1 + m_2 + \cdots + m_i = m$, de forma que $|B_1 \cup B_2 \cup \cdots \cup B_i| = m$.

Por exemplo, para calcular o valor de $\tilde{\tilde{w}}_3(v,2)$ com $m_1 + m_2 + m_3 = 2$, verificaríamos todas as possíveis combinações de m_1, m_2 e m_3 de forma que $|B_1 \cup B_2 \cup B_3| = m = 2$. Segue abaixo uma tabela com as possíveis combinações de m_1, m_2 e m_3 referentes a este exemplo.

Combinação	$\lvert B_1\rvert$	$\lvert B_2\rvert$	$\lvert B_3\rvert$	$\lvert B_1 \cup B_2 \cup B_3\rvert$
1ª	0	0	2	2
2ª	0	2	0	2
3ª	2	0	0	2
4ª	1	1	0	2
5ª	1	0	1	2
6ª	0	1	1	2

Assim seria calculado o número de arestas em um corte (B,W) em $T_{v^1} \cup T_{v^2} \cup \cdots \cup T_{v^i}$ de forma que $B = B_1 \cup B_2 \cup \cdots \cup B_i$ e que a combinação de valores de $m_1, m_2, \ldots, m_i$ leve ao número mínimo de arestas no corte.

Há no entanto um outro modo mais eficiente de fazermos isso.

Seja (B^*_{i-1}, W^*_{i-1}) um corte de largura mínima em $T_{v^1} \cup T_{v^2} \cup \cdots \cup T_{v^{i-1}}$ com $|B|^*_{i-1} = m^*_{i-1}$. Como temos que, para cada valor de m, as

funções $\tilde{\tilde{b}}_{i-1}(v,m)$ e $\tilde{\tilde{w}}_{i-1}(v,m)$ representam a largura de um corte mínimo (B^*_{i-1}, W^*_{i-1}) com $|B^*_{i-1}| = m$, $v \in B^*_{i-1}$ e $v \in W^*_{i-1}$, respectivamente, sabemos que esse corte de largura mínima vem da melhor combinação de valores de $m_1, m_2, \ldots, m_{i-1}$. Com isso, em vez de testar todas as combinações de $m_1, m_2, \ldots, m_{i-1}$ para cada subárvore em $\{T_{v^1}, T_{v^2}, \ldots, T_{v^i}\}$, testaremos apenas combinações de valores de m_i e m^*_{i-1} referentes a T_{v^i} e a $T_{v^1} \cup T_{v^2} \cup \cdots \cup T_{v^{i-1}}$, respectivamente. Isso irá gerar um corte (B, W) em $T_{v^1} \cup T_{v^2} \cup \cdots \cup T_{v^i}$ tal que $B = B^*_{i-1} \cup B_i$ e $W = W^*_{i-1} \cup W_i$, onde m_i e m^*_{i-1} formam uma melhor combinação de valores. Especificamente, as fórmulas gerais de $\tilde{\tilde{b}}_i(v,m)$ e $\tilde{\tilde{w}}_i(v,m)$ são

$$\tilde{\tilde{b}}_i(v,m) = \min_{0 \le \ell \le m} \{\tilde{\tilde{b}}_{i-1}(v,\ell) + \tilde{b}_i(v, m-\ell+1)\} \text{ e}$$
$$\tilde{\tilde{w}}_i(v,m) = \min_{0 \le \ell \le m} \{\tilde{\tilde{w}}_{i-1}(v,\ell) + \tilde{w}_i(v, m-\ell)\}.$$

Pode-se ver que em $\tilde{\tilde{b}}_i(v,m)$ temos que $m_i = m - \ell + 1$ e que $m^*_{i-1} = \ell$ e, como $B = B^*_{i-1} \cup B_i$ e $v \in B$, então $|B| = m_i + m^*_{i-1} - 1 = m$.

Na função $\tilde{\tilde{w}}_i(v,m)$, temos que $m_i = m - \ell$ e que $m^*_{i-1} = \ell$ e, como $B = B^*_{i-1} \cup B_i$ e $v \in W$, então $|B| = m_i + m^*_{i-1} = m$.

Logo, sendo k o número de filhos de v, as funções $b(v,m)$ e $w(v,m)$ são dadas por

$$b(v,m) = \tilde{\tilde{b}}_k(v,m) \quad \text{e} \quad w(v,m) = \tilde{\tilde{w}}_k(v,m).$$

Seja r a raiz de T. A largura de um corte mínimo (B, W) em T, onde $|B| = m$, é $\min\{b(r,m), w(r,m)\}$.

10 Geração de árvores binárias aleatórias

Nesta seção falaremos sobre o gerador de árvores binárias aleatórias, cuja definição foi tirada da Wikipédia [1]. Tais tipos de árvores foram utilizadas nos testes do algoritmo FST.

Árvores binárias aleatórias são árvores com um número pré-definido de vértices, construídas aleatoriamente e onde cada vértice possui no máximo dois filhos.

O gerador que utilizamos recebe um inteiro n e cria um vetor com uma permutação aleatória dos inteiros de 0 a $n-1$. Depois, constrói uma árvore binária de busca, inserindo os vértices na ordem dada pelo vetor. Isso leva tempo $O(n^2)$, dado que a inserção na árvore binária de busca pode levar tempo linear para cada um dos n vértices. No entanto, sabe-se que o tempo esperado desse processo é $O(n \lg n)$, dado que a altura esperada de uma árvore binária aleatória é $O(n \lg n)$ [2].

Usaremos a função INSERE(r, n), que criará um vértice de valor n e fará a inserção desse na árvore enraizada no vértice r.

Segue abaixo o algoritmo do gerador.

Algoritmo 6: Gerador de árvores binárias aleatórias

entrada: inteiro n

saída : raiz de uma árvore binária gerada aleatoriamente

```
   // Gera permutação aleatória
1  for i = 0 → n − 1 do
2  |  v[i] ← i;
3  end
4  for i = n − 1 → 1 do
5  |  indice ← RANDOM(i);
   |  // RANDOM(i) devolve um inteiro aleatório em {0,...,i}
6  |  v[indice] ↔ v[i];
7  end
   // Inserção dos elementos do vetor na árvore binária de busca
8  raiz ← NULL;
9  for i = 0 → n − 1 do
10 |  INSERE(raiz, v[i]);
11 end
12 return raiz;
```

11 Experimentos computacionais

11.1 Condições de teste

Os resultados mostrados a seguir foram extraídos de programas desenvolvidos em linguagem C, em uma máquina com sistema operacional *Linux Ubuntu* versão *14.04 LTS*, com kernel *Linux 3.13.0-74-generic*, sistema de operação *64 bits*, processador *Intel Core i5-4210U 1.70GHz* com *4 threads*, *7.7GB* de memória *RAM*. Para compilar os programas, foi utilizado o *gcc* versão *4.8.4* As opções de compilação usadas foram *-Wall, -ansi, -pedantic, -Wno-unused-result* e *-g*.

11.2 Árvores Binárias Aleatórias

$\|V(T)\|$	$e_T(B,W)$ **FST**/**Jansen et al.** (pior caso)	nº de vezes que encontrou o ótimo	Tempo médio **FST** (segundos)	Desvio padrão do tempo **FST**	Tempo médio **Jansen et al.** (segundos)	Desvio padrão do tempo **Jansen et al.**
100	5/2	7/10	0.000136	0.000050	0.001794	0.000818
200	7/2	9/10	0.000186	0.000089	0.008120	0.001961
400	8/3	6/10	0.000196	0.000061	0.046444	0.003297
800	8/3	3/10	0.000469	0.000091	0.350070	0.011072
1600	8/3	2/10	0.001014	0.000182	2.730767	0.054196
3200	10/3	4/10	0.001943	0.000681	22.178777	0.512264
6400	11/3	0/10	0.004158	0.000781	176.240114	2.011271
12800	10/3	0/10	0.008592	0.001632	1407.933406	45.462291
25600	12/–	–	0.024976	0.001446	–	–
51200	12/–	–	0.060082	0.003555	–	–
102400	13/–	–	0.152887	0.007932	–	–
204800	12/–	–	0.284941	0.016206	–	–
409600	13/–	–	0.619568	0.084594	–	–
819200	13/–	–	1.478659	0.093567	–	–
1638400	13/–	–	2.832498	0.147750	–	–
3276800	14/–	–	5.971384	0.199216	–	–
6553600	16/–	–	13.298427	0.831551	–	–
13107200	17/–	–	27.177690	1.280472	–	–
26214400	15/–	–	58.552827	4.089392	–	–

11.3 Árvores Ternárias

Árvores ternárias são árvores completas onde todos os nós, exceto as folhas, possuem três filhos.

Sabe-se que uma bissecção mínima numa árvore ternária completa com n vértices tem largura $\Theta(\lg n)$, e esse é o pior caso do problema bissecção mínima [4].

Altura	$\|V(T)\|$	$e_T(B,W)$ **FST**	$e_T(B,W)$ **Jansen et al.**	Tempo **FST** (segundos)	Tempo **Jansen el al.** (segundos)
1	4	2	2	0.000083	0.0000010
2	13	8	3	0.000067	0.000037
3	40	10	4	0.000098	0.000236
4	121	11	4	0.000256	0.004714
5	364	10	5	0.000503	0.070932
6	1093	11	6	0.000845	1.717452
7	3280	13	7	0.002124	45.923809
8	9841	14	8	0.006565	1235.784616
9	29524	16	–	0.021598	–
10	88573	17	–	0.073102	–
11	265720	18	–	0.250203	–
12	797161	19	–	0.815602	–

11.4 Árvores de Caminho Longo

Geramos um conjunto de árvores com diâmetro relativo alto, acima de meio. Como visto na Seção 6, todas as árvores desse tipo caem no caso 1, onde o corte da bissecção mínima é sempre no máximo dois.

$\|V(T)\|$	$e_T(B,W)$ **FST**	$e_T(B,W)$ **Jansen et al.**	Tempo **FST** (segundos)	Tempo **Jansen et al.** (segundos)
100	2	1	0.000080	0.000663
200	2	1	0.000114	0.004626
400	2	1	0.000181	0.037905
800	2	1	0.000327	0.250858
1600	2	2	0.00 584	2.136476
3200	2	2	0.001118	17.872311
6400	2	2	0.002568	130.088130
12800	2	2	0.004061	1339.149309
25600	2	–	0.008163	–
51200	2	–	0.023954	–
102400	2	–	0.042101	–
204800	2	–	0.093895	–

Para gerar as árvores ternárias e as árvores de caminho longo, foram utilizados os geradores feitos por Kampmeier [7].

Como os geradores de Kampmeier foram feitos na linguagem Java, cada teste levava bastante tempo para ser gerado, impossibilitando que obtivéssemos testes de grafos com um número de vértices maior.

11.5 Grafos de Steffen

Obtivemos instâncias reais de grafos concedidas por Steffen, feitas por Kampmeier [7], e as adaptamos para florestas para a realização dos testes com o algoritmo FST.

	$\|V(T)\|$	$e_T(B,W)$	$\Delta(T)$	Peso do corte	Tempo (segundos)
Forest 1.1	416	91	78	1354007.500000	0.003456
Forest 1.2	416	1	3	188.730896	0.004367
Forest 2.1	414	1	67	13718.629883	0.006489
Forest 2.2	414	0	3	0	0.007257
Forest 3.1	282	1	41	5570.087402	0.004447
Forest 4.2	331	1	21	76.988533	0.003482

Note que em alguns casos, o grau máximo da floresta é muito alto, e é por esse motivo que um dos resultados apresentou um número razoalvelmente grande de arestas no corte, dado que o algoritmo de FST funciona para florestas de grau limitado.

Na adaptação dos grafos para florestas, muitas das arestas foram retiradas do grafo, fazendo com que alguns vértices ficassem isolados e assim, compusessem o caminho máximo e consequentemente, em alguns casos, gerassem cortes de largura pequena.

12 Resultados

Conforme os resultados dos experimentos computacionais apresentados anteriormente, observa-se que o algoritmo de FST possui grande vantagem sobre o algoritmo de Jansen et al. em relação à complexidade de tempo e, em alguns dos casos consegue se equiparar a ele no quesito optimalidade de solução.

Vemos que ele consegue ser ótimo ou quase ótimo quando aplicado a árvores que possuem um caminho máximo longo. Isso se deve ao fato de que árvores com diâmetro grande possuem mais vértices no caminho máximo do que fora dele, e árvores desse tipo se enquadram no primeiro caso do algoritmo da dobra do diâmetro, que irá cortar no máximo duas arestas da árvore.

Nos demais casos, comparado ao algoritmo de Jansen et al., vemos que ele também tem um bom desempenho em relação a optimalidade. Isso pode ser visto nos testes com árvores ternárias completas, que são as árvores cuja bissecção mínima tem o maior número de arestas no corte, e onde o algoritmo de FST devolveu um corte de largura razoável, com razão menor que 3 em relação ao corte mínimo.

Mas, por ser um algoritmo de aproximação cuja optimalidade depende do grau máximo da árvore, ele entra em desvantagem em alguns casos de árvores de grau absurdamente grande, como pode ser visto nos resultados apresentados.

Referências

[1] Random binary tree. `https://en.wikipedia.org/wiki/Random_binary_tree`. Accessed: 2015-09-27.

[2] Thomas H. Cormen. *Introduction to algorithms*. MIT press, 2009. Theorem 12.4, Chapter 12.

[3] Cristina G. Fernandes, Tina J. Schmidt, and Anusch Taraz. On the structure of graphs with large minimum bisection. In J. Nešetřil and M. Pellegrini, editors, *The Seventh European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB)*, volume 16 of *CRM Series*, pages 291–296. Scuola Normale Superiore, 2013.

[4] Cristina G. Fernandes, Tina J. Schmidt, and Anusch Taraz. On minimum bisection and related partition problems in graphs with bounded tree width. Submitted, 2015.

[5] Michael R. Garey, David S. Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, pages 237–267, 1974.

[6] Klaus Jansen, Marek Karpinski, Andrzej Lingas, and Eike Seidel. Polynomial time approximation schemes for MAX-BISECTION on planar and geometric graphs. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2010 of *Lecture Notes in Computer Science*, pages 365–375. Springer, 2001.

[7] Alexander Kampmeier. On the computation of local and global optima for soft power diagrams. Master Thesis, 2014.

[8] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[9] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008.

[10] Tina J. Schmidt. The minimum bisection problem. PhD Thesis in preparation. Technische Universität München.

Parte II

Parte Subjetiva

13 Disciplinas relevantes para o TCC

Em geral, todas as disciplinas tiveram um papel importante para a minha formação. Mas, as que contribuíram diretamente para esse trabalho de conclusão de curso foram as seguintes:

- **Introdução à Computação** e **Princípios de Desenvolvimento de Algoritmos**

  que foram disciplinas fundamentais que me deram uma base sólida em relação à lógica de programação e desenvolvimento de algoritmos.

- **Estrutura de dados**

  onde foram vistos conceitos importantes usados nesse trabalho e também foi onde tive meu primeiro contato com grafos, em um exercício programa.

- **Análise de Algoritmos**

  onde foram estudados conceitos de classes de complexidade computacional, análise de tempo de algoritmos e programação dinâmica, que foram utilizados nesse trabalho.

- **Algoritmos em Grafos**

  onde foram vistos conceitos importantes e algoritmos base que foram muito utilizados nesse trabalho, além de ter sido uma matéria fundamental que despertou o meu interesse e curiosidade em relação à esse assunto.

14 Agradecimentos

Gostaria de agradecer, primeiramente, a minha família que me apoiou em todas as minhas decisões, sempre me incentivando a seguir em frente com meus objetivos e sonhos, e esteve ao meu lado em todos os momentos de alegrias e dificuldades.

Agradeço também as boas amizades que fiz no IME, que sempre estiveram presentes, compartilhando alegrias, conquistas, preocupações, pressões e desafios do curso e fizeram com que as longas viagens diárias da zona leste pro Butantã valessem muito mais a pena.

Sou grata a Tina Schmidt, que me permitiu estudar a sua tese de doutorado e com quem tive a oportunidade de discutir problemas relacionados ao assunto.

Por fim, também sou muitíssimo grata a todos os funcionários do IME e professores que me deram aula e aceitaram dividir um pouco do seu conhecimento. Em especial à professora Cris, que foi uma excelente orientadora, sempre muito presente, paciente e atenciosa, me auxiliando em tudo o que foi preciso para o desenvolvimento desse trabalho.