UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

TRABALHO DE CONCLUSÃO DE CURSO

Aluna: Karina Suemi Awoki

Orientadora: Cristina Gomes Fernandes

Sumário

1 Introdução

O assunto abordado no Trabalho de Conclusão de Curso 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 $B \cup W = V(G)$ e $B \cap W = \emptyset$. 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 é NP-difícil [3] e a melhor aproximação conhecida para o caso geral do problema tem razão $O(\lg n)$ [6], onde n é o número de vértices do grafo. Por outro lado, sabe-se que, para árvores e grafos que de uma certa maneira se assemelham a árvores, há um algoritmo polinomial de programação dinâmica para encontrar uma bissecção mínima, proposto por Jansen, Karpinski, Lingas e Seidel [4].

Fernandes, Schmidt e Taraz têm interesse em entender a estrutura dos grafos cuja bissecção mínima tem largura grande, de modo a desenvolver bons algoritmos de aproximação para o problema ou identificar melhor as classes de grafos onde o problema torna-se especialmente difícil. Para tanto, eles têm feito estudos de certas questões em árvores, em grafos que têm uma estrutura semelhante às árvores, e também em grafos planares [1, 2], para os quais a complexidade do problema encontra-se em aberto.

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

2 Objetivos

O principal objetivo deste trabalho foi o estudo, implementação e análise de um algoritmo recente, proposto por Fernandes, Schmidt e Taraz [1], para encontrar uma bissecção aproximadamente mínima em árvores de grau limitado, denotaremos esse algoritmo por FST. A vantagem deste algoritmo sobre o algoritmo de Jansen et al. [4], 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.

Como já mencionado, o algoritmo de Jansen et al. baseia-se em programação dinâmica. Já o algoritmo de Fernandes et al., para atingir um consumo linear, 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. É um algoritmo mais refinado, dividido na implementação de uma série de etapas menores.

Uma vez implementados os dois algoritmos, foi feita uma análise da sua performance em árvores binárias geradas aleatoriamente, e em árvores ternárias. Obtivemos instâncias reais de um problema relacionado e as utilizamos também no estudo experimental do algoritmo implementado.

3 Método

O estudo do algoritmo de Fernandes et al. seguiu um roteiro de como a implementação foi feita, conforme a proposta do TCC. A linguagem escolhida para a implementação foi `C` e os algoritmos implementados foram colocados na página do gitHub. 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 [7] 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 a implementação obtida 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, e a parte da implementação que estava pronta foi apresentada a ela, e algumas discussões foram feitas e em especial, a Tina conseguiu um conjunto de instâncias reais com [5] para usarmos em nossos experimentos.

Em paralelo ao estudo e desenvolvimento da implementação, foram estudados também dois artigos da literatura. São eles o artigo de Garey, Johnson e Stockmeyer [3], que contém a prova de que o problema é NP-difícil, e o artigo de Jansen et al., que apresenta o algoritmo de programação dinâmica para o problema em árvores. Após a implementação do algoritmo de Fernandes et al., foi feita a implementação do algoritmo de Jansen et al. e a comparação dos dois.

4 Definições

4.1 Conjunto $[n]$

Denotemos por $\{1, 2, \dots\ n\}$ o conjunto $[n]$.

4.2 Conjunto de vértices e arestas

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

4.3 Grau de vértices

Em um grafo G, o número de arestas que incidem em um determinado vértice v é chamado de **grau** de v e será representado por $grau_G(v)$. O **grau máximo**, que é o grau de um vértice de maior grau em G, será representado por $\Delta(G)$.

4.4 Caminhos em árvores

Um **caminho** em uma árvore T, de comprimento p, é uma sequência de vértices $v_0, v_1, \dots, v_{p-1}, v_p$ onde v_{k-1}, v_k é uma aresta para todo $k = 1, 2, \dots, p$. Como existe um único caminho que conecta quaisquer dois vértices v_0 e v_p em T, chamaremos tal caminho em T de v_0,v_p-**caminho**.

4.5 Distâncias entre vértices

A **distância** entre dois vértices a e b de T é representada por $dist(a, b)$, e é igual ao comprimento do a,b-caminho em T.

5 Caminho máximo em árvores

5.1 Algoritmo que encontra um caminho máximo

Um **caminho máximo** em uma árvore T é um caminho em T de comprimento máximo.

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. Depois, repetimos o mesmo processo a partir de y_0, encontrando um vértice x_0 mais distante de y_0. Para encontrar um vértice mais distante de algum vértice dado, basta usar uma busca em largura. Feito isso, temos um caminho máximo em T: o único caminho que liga x_0 a y_0.

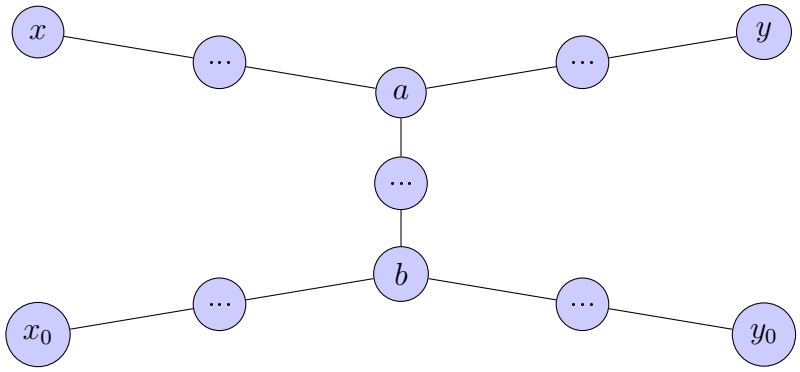
5.2 Prova de funcionalidade do algoritmo

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

Demonstração. Suponhamos 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, de comprimento $c \geq 1$, que conecta os dois caminhos, possuindo assim apenas o vértice a no x,y-caminho e apenas o vértice b no x_0,y_0-caminho.



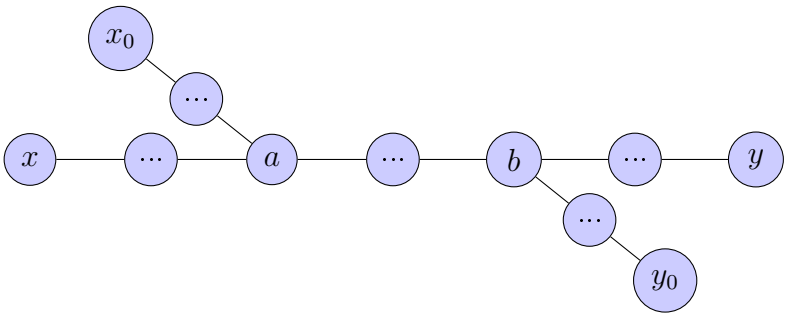
Portanto, temos que $dist(x, y) = dist(x, a) + dist(a, y)$ e $dist(x_0, y_0) = dist(x_0, b) + dist(b, y_0)$.

Como o x,y-caminho é máximo, $dist(x, y) \geq dist(x_0, y)$, e isso implica que $dist(x, a) \geq dist(x_0, b) + c$.

Como x_0 é um vértice mais distante de y_0, temos que $dist(x_0, y_0) \geq dist(x, y_0)$, que implica que $dist(x_0, b) \geq dist(x, a) + c$. Logo temos que $dist(x, a) \geq dist(x, a) + 2c$, 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 $dist(x, a) \leq dist(x, b)$ e que $dist(x_0, a) \leq dist(x_0, b)$, como na figura abaixo. (Do contrário troque a por b e possivelmente x por y.)



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

  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. Se $v' \neq b$, então, como y_0 é um vértice mais distante de v,

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

  o que implica que $dist(v', y_0) = dist(v', y)$ assim como $dist(b, y) = dist(b, y_0)$. Ou seja, $dist(x, y) = dist(x_0, y_0)$. Resta

analisar o caso em que $v' = b$ (é um caso especial, dado que pode-se ter arestas em comum entre v,v'-caminho e o y,b-caminho). Como y é um vértice mais distante de x, temos que $dist(v', y_0) \leq dist(v', y)$. Como x_0 é um vértice mais distante de y_0, temos que $dist(x_0, v') \geq dist(y, v')$.

Finalmente, como y_0 é um vértice mais distante de v, vale que $dist(y_0, v') \geq dist(x_0, v')$.

Dessas três desigualdades, concluímos que $dist(y_0, v') = dist(y, v') = dist(x_0, v') = dist(x, v')$, onde a última igualdade vale pois já mostramos que $dist(x_0, a) = dist(x, a)$.

Assim sendo $dist(x_0, y_0) = dist(x, y)$, completando a prova.

□

5.3 Diâmetro de um Grafo

Dizemos que o **diâmetro** de um grafo conexo G é o comprimento de um caminho máximo de G. Em outras palavras, o diâmetro pode ser definido como:

$$diam(G) = \max\{dist(x, y) : x, y \in V(G)\}.$$

Seja G um grafo não necessariamente conexo, o **diâmetro relativo** de G é a razão entre o número de vértices de um caminho máximo e o número total de vértices de G. Este pode também ser representado por

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

Nota-se que para todo grafo G com $V(G) \neq \emptyset$, temos que $0 < diam^*(G) \leq 1$, pois haverá no mínimo um vértice de G no caminho máximo, e no máximo, todos os vértices de G.

6 Lemas de cortes aproximados

Em [7], são apresentados três lemas e seus respectivos algoritmos, que, dada uma floresta G e um inteiro $0 < m \leq n$, onde n é o número de vértices de G, produzem cortes com algumas propriedades.

Abaixo reproduzimos os três resultados de [7] que atestam as propriedades dos cortes produzidos por estes algoritmos:

6.1 Lema do corte aproximado simples para árvores

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 e r um vértice de T. Assumiremos que a função CALCULA_NUMERO_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. Nota-se que essa função pode ser executada usando uma busca em profundidade e toma tempo $O(n)$.

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

Algorithm 1: Computa corte aproximado em uma árvore

input : árvore $T = (V, E)$ com n vértices, $m \in [n]$ e uma raíz r
output: Um corte (B, W) tal que $\frac{m}{2} < |B| \leq m$

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

Análise do Algoritmo

Suponha que $T = (V, E)$. É fácil ver que se $m = n$, então os valores $B = V$ e $W = \emptyset$ satisfazem as condições do lema, dado que $e_G(B, W)$ será 0. Isso pode ser computado em tempo $O(n)$.

Caso contrário, é necessário escolher uma raiz arbitrária r para T e calcular o número de vértices das sub-árvores enraizadas em cada um dos nós, que é o número de descendentes do nó. Isso serve para que saibamos a quantidade de vértices das sub-árvores sem que precisemos percorrer todos os nós das sub-árvores a cada consulta.

Feito isso, temos que v é o vértice analisado no momento e $V_1, V_2, \ldots, V_k$ são as sub-árvores enraizadas nos k filhos de v. Se algum V_i satisfi-

zer $\frac{m}{2} < |V_i| \leq m$, podemos retornar a sub-árvore $|V_i|$ como sendo o conjunto B, satisfazendo o Lema 2, dado que o corte será 1. Caso contrário, se $|V_i| > m$ para algum i, assumimos que v agora é a raiz de V_i e aplicamos novamente esse processo em v.

Nota-se que esse procedimento irá parar em algum momento, dado que a árvore é finita, e ele parará quando encontrar uma sub-árvore que satisfaça o Lema 2 (e essa sub-árvore será o conjunto B) ou quando chegarmos numa situação em que $|V_i| \leq \frac{m}{2}$ para todos os k filhos de v. Nesse último caso, sabemos que $|V_1 \cup V_2 \cup \ldots \cup V_k| \geq m$, pois a sub-árvore enraizada em v possui mais que m vértices. Sabemos também que $|V_i| \leq \frac{m}{2}$ para todos os k filhos de v. Com isso, percorremos os V_i em ordem, e adicionamos V_i ao conjunto B, parando antes que $|B| > m$. Isso nos dará $e_T(B, V \backslash B) < grau_T(v) \leq \Delta(T)$, o que satisfaz o lema. Dado que $|V_j| \leq \frac{m}{2}$ para todo j, sabemos que existe um i tal que $\frac{m}{2} < |V_1 \cup V_2 \cup \cdots \cup V_i| \leq m$, pois a união das sub-árvores enraizadas em v possui mais que m vértices e, para todo ℓ que satisfaça $|V_1 \cup V_2 \cup \cdots \cup V_\ell| \leq \frac{m}{2}$, temos que $|V_1 \cup V_2 \cup \cdots \cup V_\ell \cup V_{\ell+1}| \leq m$.

6.2 Lema do corte aproximado simples para florestas

Lema 3 ([7, Lemma 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 $O(n)$.*

Agora examinaremos como computar o corte descrito pelo lema. Nota-se que essa é uma adaptação do algoritmo anterior, trocando árvore por floresta.

Algorithm 2: Computa corte aproximado simples em uma floresta

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

1 $B \leftarrow \emptyset$;
2 Sejam $V_1, V_2, \ldots, V_k$ os conjuntos de vértices das componentes conexas de G;
3 **for** $i = 1 \rightarrow k$ **do**
4 **if** $|V_i| + |B| \leq m$ **then**
5 $B \leftarrow B \cup V_i$;
6 **end**
7 **else if** $|B| < m$ **then**
8 $(B', W') \leftarrow Algoritmo1(G[V_i], |V_i|, m - |B|)$;
  // Devolve o corte (B', W') em $G[V_i]$ com $\frac{m-|B|}{2} < |B'| \leq m - |B|$ (usando o Lema 2)
9 $B \leftarrow B \cup B'$;
10 break;
11 **end**
12 **end**
13 **return** $(B, V \setminus B)$;

Análise do Algoritmo

Temos que $T_1, T_2, \ldots, T_k$ são as árvores de G. Primeiro percorremos as árvores na ordem que compõem. Seja ℓ o maior inteiro tal que $s = \sum_{i=1}^{\ell} |V(T_i)| \leq m$. Colocamos $T_1, T_2, \ldots, T_\ell$ no conjunto B. Caso $s = m$, temos $|B| = m$ com $e_G(B, V \setminus B) = 0$, o que satisfaz o lema. Caso contrário, ao encontrar um corte (B', W') em $T_{\ell+1}$ que satisfaça $\frac{m-s}{2} < |B'| \leq m - s$ com $e_{T_{\ell+1}}(B', W') \leq \Delta(T_{\ell+1})$, e acrescentar B' ao conjunto B, teremos $\frac{m+s}{2} < |B| \leq m$ com $e_G(B, W) = \Delta(T_{\ell+1}) \leq \Delta(G)$. Logo, depois de acrescentar B' em B, teremos o corte $(B, V \setminus B)$ que satisfaz o lema, e podemos fazer isso utilizando o Lema 2.

Nota-se que, como a linha 2 pode ser feita usando uma busca em profundidade e as demais linhas envolvem verificar o tamanho das componentes (que já foi calculado na linha 2) e uma chamada ao Algoritmo 1, então essa operação pode ser feita em tempo $O(n)$, dado que todas as operações citadas são executadas em tempo linear.

6.3 Lema do corte aproximado

Lema 4 ([7, Lemma 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 $O\left(\left\lceil \frac{2c}{1-c} \right\rceil n\right)$.*

Algorithm 3: Computa corte aproximado em uma floresta

input : floresta $G = (V,E)$ com n vértices, $m \in [n]$, $c \in (\frac{1}{2}, 1)$
output: Um corte (B,W) tal que $cm \leq |B| \leq m$

if $c \leq \frac{1}{2}$ **then**
  $(B,W) \leftarrow$ Algoritmo 2(G,n,m) ;
  // Devolve o corte (B,W) em G com $\frac{m}{2} < |B'| \leq m$ (usando o Lema 3)
  $B \leftarrow B'$;
else
  $B \leftarrow \emptyset$;
  while $|B| < cm$ **do**
    $(B',W') \leftarrow$ Algoritmo2$(G[V \setminus B], n-|B|, m-|B|)$;
    // Devolve o corte (B',W') em $G[V \setminus B]$ com $\frac{m-|B|}{2} < |B'| \leq m-|B|$ (usando o Lema 3)
    $B \leftarrow B \cup B'$;
  end
end
return $(B, V \setminus B)$

Análise do Algoritmo

Sabemos que se $c \leq \frac{1}{2}$, podemos aplicar o Algoritmo 2, dado que o resultado nos dará um conjunto B' tal que $|B'| > \frac{m}{2} \geq cm$ e $e_G(B,W) \leq \Delta(G)$.

Caso contrário, temos que $B = \emptyset$. Executaremos o Algoritmo 2, referente ao Lema 3 diversas vezes para encontrar um corte (B',W') em $G[V \setminus B]$ tal que $\frac{m-|B|}{2} < |B'| \leq m-|B|$, acrescentando o conjunto B' a B em cada uma das chamadas ao Lema 3. Fazemos isso até que $|B| \geq cm$ seja

satisfeito. Nota-se que em algum momento $|B| \geq cm$ irá ocorrer, pois a cada vez que executamos o Algoritmo 2, acrescentamos pelo menos um vértice em B, e $|B| > m$ nunca irá ocorrer, pois o conjunto B' obtido no Lema 3 é tal que $|B'| \leq m - |B|$. Logo, serão adicionados no máximo $m - |B|$ vértices em B.

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

Caso $c \leq \frac{1}{2}$, executamos o Algoritmo 2 uma única vez, portanto o tempo é $O(n)$, que equivale a $O(\left\lceil \frac{2c}{1-c} \right\rceil n)$. Em relação a largura do corte, será devolvido o corte do Algoritmo 2 sem nenhuma modificação, então, sabe-se que $e_G(B, W) \leq \Delta(G)$. Provaremos agora que $\Delta(G) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$. Como $0 < c < 1$, temos que $0 < 1 - c$ e $0 < 2c$, implicando em $0 < \frac{2c}{1-c}$, assim sendo, $1 \leq \left\lceil \frac{2c}{1-c} \right\rceil$. Portanto, como $\Delta(G) \geq 0$, então temos que $\Delta(G) \leq \left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$, satisfazendo assim as propriedades do Lema 4.

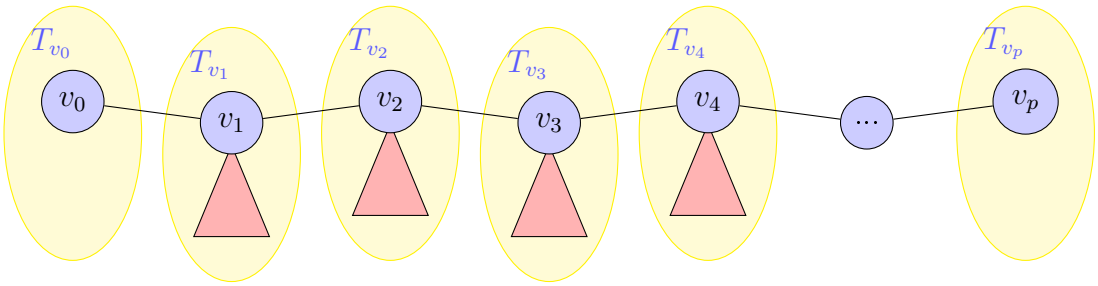
Caso contrário, $c > \frac{1}{2}$. Como em todas as vezes que executamos a linha 7 do Algoritmo 3, vale que $|B| < cm$, então também é válido que $m - |B| > (1 - c)m$. E como $m - |B|$ é o valor do m que passaremos para o Algoritmo 2, sabemos que o conjunto B do corte devolvido terá mais que $\frac{m - |B|}{2}$ 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, dando, no total, um tempo $O(\left\lceil \frac{2c}{1-c} \right\rceil n)$. Usaremos essa mesma linha de pensamento para calcular o largura máxima do corte devolvido. Sabemos que o Algoritmo 2 será chamado $\left\lceil \frac{2c}{1-c} \right\rceil$ vezes, e para cada uma das vezes ele devolve um corte cuja largura máxima é $\Delta(G)$, portanto, a largura máxima do corte devolvido é $\left\lceil \frac{2c}{1-c} \right\rceil \Delta(G)$.

7 Rotulação dos vértices da árvore

Mais adiante, iremos nos referir a vértices com uma característica específica, e para isso precisamos atribuir um rótulo numérico $x \in [n]$ distinto para cada um dos vértices da árvore, onde n é o número de vértices da árvore T. Denotaremos o **rótulo** de um vértice v por $rot(v)$.

Primeiramente, usaremos o método mencionado na seção 5 paracalcular o v_0-v_k-caminho, de comprimento p, que é máximo em T. Para todo vértice $v \in V(v_0\text{-}v_k\text{-caminho})$, teremos uma árvore $T_v \in G \setminus E(v_0\text{-}v_k\text{-caminho})$ de modo que $v \in T_v$.

Dessa forma, teremos sub-árvores enraizadas nos vértices do caminho máximo, como o mostrado na figura abaixo.



Temos que cada vértice contido no v_0-v_p-caminho receberá uma rotulação de forma que:

- $rot(v_i) > rot(v_j)$ para todo $i < j$ e
- $rot(v_{i-1}) < v' \leq rot(v_i)$ para todo vértice v' contido na sub-árvore enraizada em v_i.

7.1 Descrição do algoritmo de rotulação

Essa rotulação pode ser obtida facilmente manipulando a lista de adjacência e em seguida, usando uma busca em profundidade. Ambas as ações são executadas em tempo linear, então pode-se rotular uma árvore em tempo $O(n)$.

Para cada vértice v_i contido no v_1-v_p-caminho, percorremos a sua lista de adjacência e quando encontrarmos o vértice v_{i-1}, colocamos ele no início da lista. Dessa forma, quando aplicamos uma busca em profundidade nessa árvore, usando o v_p como raiz, a busca descerá primeiro pelos vértices do

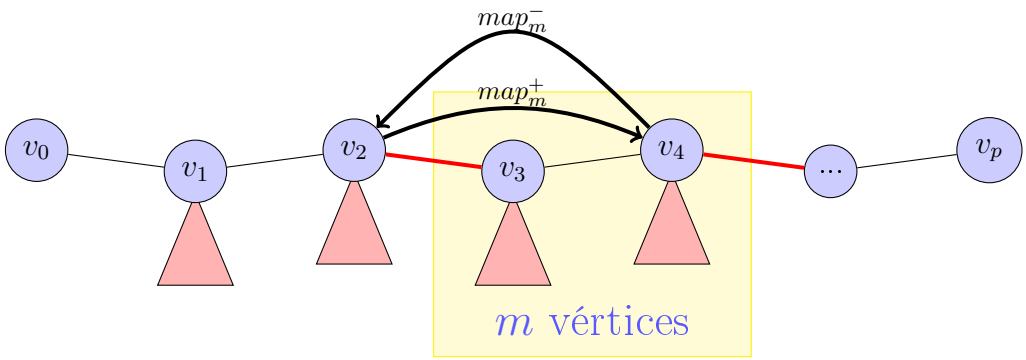
caminho máximo e atribuirá a rotulação somente quando terminar de ver todos os vértices filhos, garantindo assim que para todo $r \in \{1, \ldots, p\}$, temos que $rot(T_{v_i}) \leq rot(v_i)$ e $rot(v_i) > rot(v_{i-1})$.

7.2 Mapeamento dos rótulos de vértices

Vamos definir agora as funções que mapeiam um vértice em outro, utilizando a rotulação como base. Chamaremos de map_m^+ a função que devolve o m-ésimo próximo vértice e, de map_m^- a função inversa da primeira, sendo assim ela devolve o m-ésimo vértice anterior. Dessa forma, podemos ver que, seja v um vértice, $map_m^+(rot(v)) = (rot(v) + m) \bmod n$ e $map_m^-(rot(v)) = (rot(v) - m) \bmod n$.

Para facilitar, identificaremos os vértices pelos seus rótulos. Sendo assim, temos que para um vértice v, $map_m^+(v) = (v + m) \bmod n$ e $map_m^-(v) = (v - m) \bmod n$ como uma simplificação das funções enunciadas anteriormente.

Nota-se que para todo $m \in [n]$, caso $n > 2$, se existe um v tal que $v \in v_0$-v_k-caminho e $map_m^+(v) \in v_0$-v_k-caminho, então existe um corte $(B, W) \in T$, tal que $B = \{v+1, v+2, \ldots, v+m\}$, com $|B| = m$ e $e_T(B, W) \leq 2 \leq \Delta(T)$. Isso pode ser visto na figura abaixo.



Caso contrário, $n \leq 2$. É fácil ver que todos os vértices da árvore estarão no caminho máximo, portanto, para qualquer $m \in [n]$, teremos um conjunto B tal que $|B| = m$ e $e_T(B, W) \leq |E(T)| = n - 1 = \Delta(T) \leq 2$.

7.3 Lema do corte em árvores de caminho longo

Futuramente usaremos os resultados do lema abaixo na prova de um dos teoremas.

Lema 5 ([7, Lemma 5]). *Para toda árvore T com n vértices, $diam^*(T) > \frac{1}{2}$ 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 $O(n)$.*

Podemos ver que a função $map_m^+(x)$ é injetora, dado que numa soma, elementos distintos do domínio tem imagens distintas.

Uma árvore com a propriedade $diam^*(T) > \frac{1}{2}$ é uma árvore cujo caminho máximo contém mais que a metade dos vértices da árvore, o que nos leva ao fato de que existem mais vértices no caminho máximo do que fora dele.

Como o mostrado anteriormente, se um vértice v mapeia um vértice v' e ambos estão no caminho máximo, então existe um corte (B, W) em T com $e_T(B, W) \leq 2$. Dado que $diam^*(T) > \frac{1}{2}$, e usando o fato de que $map_m^+(x)$ é uma função injetora e que existem mais vértices no caminho máximo do que fora dele, temos que pelo menos um vértice do caminho máximo irá mapear outro vértice do caminho máximo. Isso ocorre porque todos os vértices mapeiam algum outro vértice, e não temos um número suficiente de vértices fora do caminho máximo para serem mapeados por todos os vértices do caminho máximo, então algum vértice do caminho máximo vai acabar mapeando outro do caminho máximo, e assim, obtemos um corte (B, W), com $e_T(B, W) \leq 2$, como o descrito na subseção anterior.

Um algoritmo que encontra corte desse tipo, em árvores com essa propriedade, irá percorrer os vértices do caminho máximo e verificar se algum deles mapeia um vértice do caminho máximo. Isso pode ser feito em tempo $O(n)$ se for mantido um vetor binário que indica se cada um dos vértices está ou não no caminho máximo.

8 Algoritmo da dobra do diâmetro

8.1 Teorema da dobra do diâmetro

Teorema 1 ([7, Theorem 4]). *Para toda árvore T com $n \geq 3$ vértices e todo $m \in [n]$, o conjunto de vértices de T pode ser particionado em três partes B, W e S tal 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* $0 < |S| \leq \frac{n}{2}$, $e_T(B, W, S) \leq \frac{2 \cdot \Delta(T)}{diam^*(T)}$, *e* $diam^*(T[S]) \geq 2diam^*(T)$.

Uma partição (B, W, S) que satisfaz 1 ou 2 pode ser computada em tempo $O\Big(\frac{n}{diam^*(T)}\Big)$

8.2 Algortimo da dobra do diâmetro

Algorithm 4: Inserção do nó n na árvore binária de busca

input : árvore T com n vértices e $m \in [n]$
output:

9 Algoritmo FST para bissecção

9.1 Teorema do corte exato

Lema 6. *Para toda árvore T com $n > 2$ vértices e toda aresta $e = \{v_1, v_2\}$ em T, tal que $v_1, v_2 \in V$ e $grau_T(v_1) = grau_T(v_2) = 1$, temos que $\Delta(T) = \Delta(T \cup e)$.*

Demonstração. Como $n > 2$ vértices, então temos que $\Delta(T) \geq 2$. Sabemos que ao adicionar e em T, o grau de v_1 e v_2 passará a ser 2, e os demais vértices não terão o grau alterado.

Seja v um vértice de T tal que $grau_T(v) = \Delta(T) \leq 2$, dado que $grau_T(v_1) = grau_T(v_2) = 1$ sabe-se que $v \neq v_1$ e $v \neq v_2$, o que leva a $grau_{T \cup e}(v) = grau_T(v) = \Delta(T)$.

Dessa forma, temos que

$$\begin{aligned}
\Delta(T \cup e) &= \max\{grau_{T \cup e}(v_1), grau_{T \cup e}(v)\} \\
&= \max\{2, grau_{T \cup e}(v)\} \\
&= grau_{T \cup e}(v) \\
&= grau_T(v) \\
&= \Delta(T)
\end{aligned}$$

□

Teorema 2 (Teorema do corte exato [7, Theorem 6])**.** *Para todo $i \in \mathbb{N}_*^+$, toda árvore T com n vértices e $diam^*(T) > \frac{1}{2^i}$, e todo $m \in [n]$, existe um corte (B, W) em T com $|B| = m$ e $e_T(B, W) \leq 4 \cdot 2^i \cdot \Delta(T)$. Um corte que satisfaz esses requisitos pode ser computado em tempo $O\Big(\frac{n}{diam^*(T)}\Big)$.*

Demonstração. Utilizaremos indução para provar o Teorema 2 (não levaremos em conta o tempo de execução agora).

Primeiramente, temos que $diam^*(T) \leq 1$, como o mostrado na seção 5.3. Sendo assim, sabe-se que $i \geq 1$, logo $i = 1$ será a base utilizada na indução.

Caso base, $i = 1$. Para todo $n \leq 2$, sabe-se que $|E(T)| = \Delta(T)$, e nesse caso, $e_T(B, W) \leq |E(T)| = \Delta(T) \leq 4 \cdot 2^i \cdot \Delta(T)$. Para $n > 2$ temos que $diam^*(T) > \frac{1}{2}$, logo, podemos utilizar o Lema 5 para obter um

corte (B, W) com $e_T(B, W) \leq 2 < 4 \cdot 2^i \cdot \Delta(T)$. Portanto, esse teorema é válido para $i = 1$.

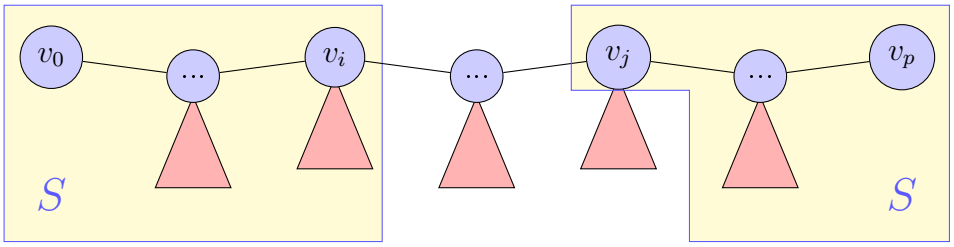
Caso $i \geq 2$. Provaremos que esse teorema vale para i, assumindo que ele é válido para $i-1$. De modo semelhante ao caso anterior, para $n \leq 2$, temos que $e_T(B, W) \leq |E(T)| = \Delta(T) \leq 2 \cdot 2^i \cdot \Delta(T)$. Porém, se $n > 2$, podemos utilizar o Teorema 1, que nos devolverá cortes do tipo 1 ou 2.

- Se for do tipo 1, teremos $|B| = m$ e $e_T(B, W) \leq 2 \leq 2 \cdot 2^i \cdot \Delta(T)$.
- Se for do tipo 2, podemos obter um corte (B', W', S') tal que $|B'| = m$ ou $|B'| < m \leq |B'| + |S'|$ com $e_T(B', W', S') \leq \dfrac{2 \cdot \Delta(T)}{diam^*(T)} \leq 2 \cdot 2^i \cdot \Delta(T)$.

  Se $|B'| = m$, temos então as propriedades do corte desse teorema.

  Caso contrário, $|B'| < m \leq |B'| + |S'|$, o que nos leva a $0 < m - |B'| \leq |S'|$. Pelas propriedades do Teorema 1, sabemos que $diam^*(T[S']) \geq 2 \cdot diam^*(T) > \dfrac{1}{2^{i-1}}$. Como $0 < m - |B'| \leq |S'|$ e $diam^*(T[S']) > \dfrac{1}{2^{i-1}}$, podemos usar o Teorema 2 em $T[S]$ com m valendo $m - |B'|$ e i valendo $i - 1$.

  Porém, existem casos onde o conjunto S' retornado pelo Teorema 1 não é conexo, pois, de acordo com o Algoritmo 4, S' pode se dividir em duas componentes, uma no começo e outra no fim do caminho máximo, como o mostrado na figura abaixo.



  Como o Teorema 2 se trata de árvores, precisamos tornar S' conexo de forma a não alterar o $\Delta(T)$. Sabemos que se adicionarmos uma aresta e em T, que liga o primeiro ao último vértice do caminho máximo, então S' será conexo. Diremos que $T' = T \cup e$.

  Nota-se também que em ambas as componentes de S', temos que o caminho máximo de cada uma é a interseção entre o caminho máximo de T e a componente. Logo, se adicionando e, teremos um caminho máximo em S' e preservaremos, em parte, o caminho máximo de T. Além

disso, de acordo com o Lema 6, teremos que $\Delta(T) = \Delta(T')$, já que e conectará dois vértices extremos do caminho máximo, que possuem grau um. Podemos, então, passar a árvore $T'[S']$ para o Teorema 2, sem nos preocupar com a modificação do grau máximo.

Como assumimos que o Teorema 2 vale para $i-1$, então existe um corte (B'', W'') 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, sabemos que existe um corte (B, W) tal que $B = B' \cup B''$ e $W = V(T) \setminus B$, sendo que $|B| = |B'| + m - |B'| = m$ e

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

provando assim que o Teorema 2 é válido para i partindo do pressuposto de que ele vale para $i-1$. Logo, temos que o Teorema 2 é válido para todo $i \in \mathbb{N}_*^+$.

□

9.2 Algoritmo do corte exato

Seja $T = (V, E)$ uma árvore e $v_1, v_2 \in V$. Assumimos que a função COMPUTA_CAMINHO_MÁXIMO(T) devolve um vetor contendo os vértices do caminho máximo, de acordo com a ordem que aparecem no caminho, isso pode ser feito em tempo $O(n)$, como o descrito na seção 5.

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

Algorithm 5: Computa corte exato em uma árvore

input: árvore $T = (V, E)$ com n vértices e $m \in [n]$
outp : corte (B, W) tal que $|B| = m$

```
1 B ← ∅;
2 S' ← V;
3 raiz ← um vértice arbitrário de T;
4 while |B| < m do
5 |   P ← COMPUTA_CAMINHO_MAXIMO(T[S']);
6 |   [(B', W', S'), raiz] ← Algoritmo4(T[S'], |S'|, m − |B|, P, raiz);
7 |   B ← B ∪ B';
8 end
9 return (B, V \ B);
```

Análise do Algoritmo

No algoritmo, temos que transformar o conjunto de vértices S' em um grafo. Poderíamos fazer isso verificando todas as arestas que ligam dois vértices de S' e colocar no grafo, mas podemos fazer isso no Algoritmo 4 de forma mais eficiente. Sabemos que no Algoritmo 4, analisamos os vértices de S' que se ligam a vértices de $V(T) \setminus S'$. Portanto, podemos usar uma marcação para indicar que as arestas que ligam S' e $V(T) \setminus S$ estão inativas, tornando S' uma componente conexa isolada. Isso pode ser feito em tempo $O(1)$. O Algoritmo 4 também devolverá algum vértice de S' que será usado como raiz na próxima interação do Algoritmo 4.

Acrescentar a aresta e, como o descrito no Teorema 2 também pode ser feito de forma mais simples no Algoritmo 4, que assim como

10 Complexidade do problema da bissecção

Nesta seção veremos duas reduções que provam que o problema da bissecção mínima é NP-Completo, assumindo que o problema Max Sat2 é NP-completo.

Primeiramente, vamos definir alguns problemas para facilitar a compreensão das reduções.

Problema 1 (Max Sat2 - Satisfatibilidade máxima com no máximo duas literais por cláusula [3]). *Dadas p cláusulas na forma disjuntiva distintas, com no máximo duas literais cada, e um inteiro $k \leq p$, encontrar valores para cada uma das literais, de forma que k ou mais variáveis sejam verdadeiras.*

Problema 2 (Simple Max Cut - Corte Máximo Simples). *Dado um grafo $G(V, E)$, onde cada uma das arestas tem peso 1, e um inteiro positivo $W \leq |E|$, encontrar um corte (T, F) em T tal que $e_G(T, F) \geq W$.*

10.1 Redução Max Sat2 para SimpleMaxCut

Temos um problema Max Sat2 e precisamos transformar a sua solução na solução do problema Simple Max Cut. Para isso vamos definir alguns conceitos referentes ao problema Max Sat2:

- $p =$ número de cláusulas
- $n =$ número de variáveis

Primeiramente, vamos definir essa redução para depois explicar o porquê ela funciona. A redução consiste em criar um grafo $G = (V, E)$, de acordo com uma entrada para o problema Max Sat2. Segue o conjunto de vértices e arestas de G, respectivamente.

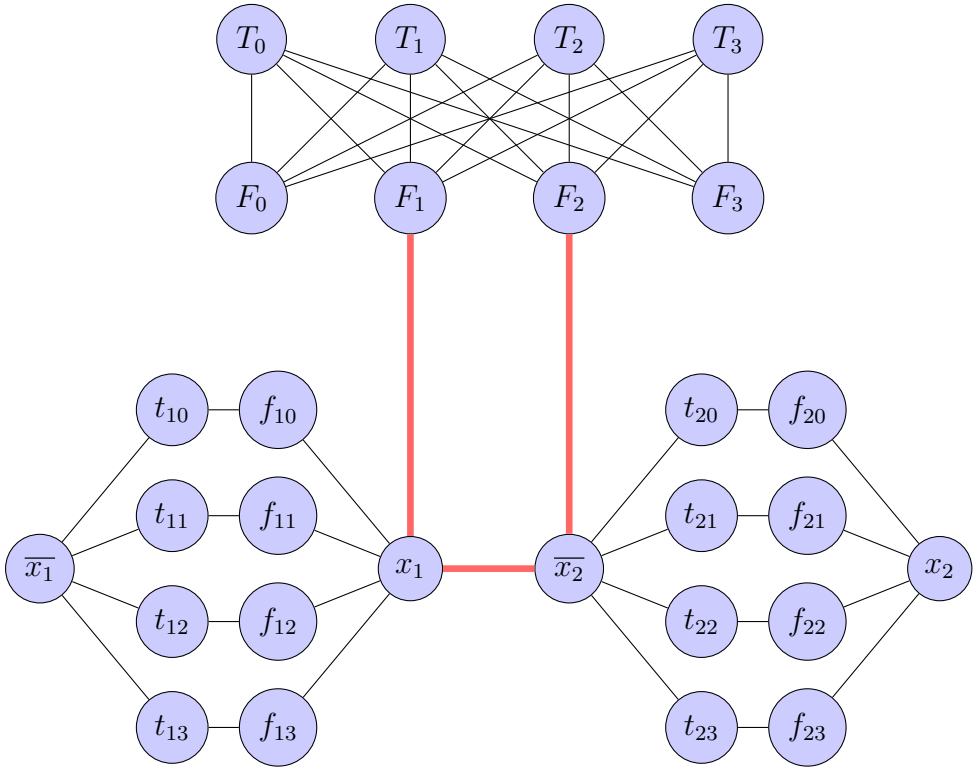
$$
\begin{aligned}
V = &\{x_i : 1 \leq i \leq n\} && \cup \\
&\{\overline{x_i} : 1 \leq i \leq n\} && \cup \\
&\{T_j : 0 \leq j \leq 3p\} && \cup \\
&\{F_j : 0 \leq j \leq 3p\} && \cup \\
&\{t_{ij} : 1 \leq i \leq n, 0 \leq j \leq 3p\} && \cup \\
&\{f_{ij} : 1 \leq i \leq n, 0 \leq j \leq 3p\} && \cup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

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

E para cada uma das p cláusulas da forma $C_i = (a_i \vee b_i)$, temos também as arestas a seguir.

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

Dessa forma, temos o grafo $G = (V, E)$, com $E = E_1 \cup E_2$. Segue abaixo um exemplo de como ficaria G sendo gerado por uma cláusula do tipo $(x_1 \vee \overline{x_2})$. As arestas de E_2 estão representadas pela cor **vermelha**.



Se encontrarmos um corte (T, F) em G tal que $e_G(T, F) \geq |A_1| + 2 \cdot k$, então teremos que, para todo vértice do tipo x_i, se $x_i \in T$, a variável que ele representa será verdadeira, já se $x_i \in F$, a variável que ele representa será falsa. Tendo que as variáveis possuem os valores conforme o que foi dito anteriormente, k ou mais cláusulas serão verdadeiras.

Explicação:

Mostraremos agora que para todas as entradas do problema Max Sat2, existe uma solução da sua redução para o problema Simple Max Cut. Para isso, vamos supor que já temos um conjunto de valores para as variáveis do problema Max Sat2 de forma que k ou mais cláusulas sejam verdadeiras.

Sabemos que existem no máximo $2p$ arestas do tipo $\{x_i, F_j\}$ ou $\{\overline{x_i}, F_j\}$. Dado que $2p < 3p+1$, que as funções $f(i) = 2 \cdot i - 1$ e $g(i) = 2 \cdot i$ têm imagem ímpar e par, respectivamente, e ambas são injetoras, portanto, temos que cada F_i se liga a apenas um vértice do tipo x_i ou $\overline{x_i}$.

Definiremos agora os vértices contidos nos subconjuntos T e F de G. Suponhamos que 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 pertence ao mesmo subconjunto que t_{jk} e ambos não pertencem ao mesmo subconjunto que $\overline{x_j}$ e que f_{jk}. Assim sendo, temos que todas as arestas de E_1 estão no corte (T, F). Suponhamos também que se a variável x_i é verdadeira, então, o vértice x_i está em T, e se mantermos as propriedades que foram citadas anteriormente, isso não afetará o fato de todas as arestas de E_1 estarem no corte (T, F). Portanto, assumiremos inicialmente que essa é a disposição dos vértices nos subconjuntos T e F.

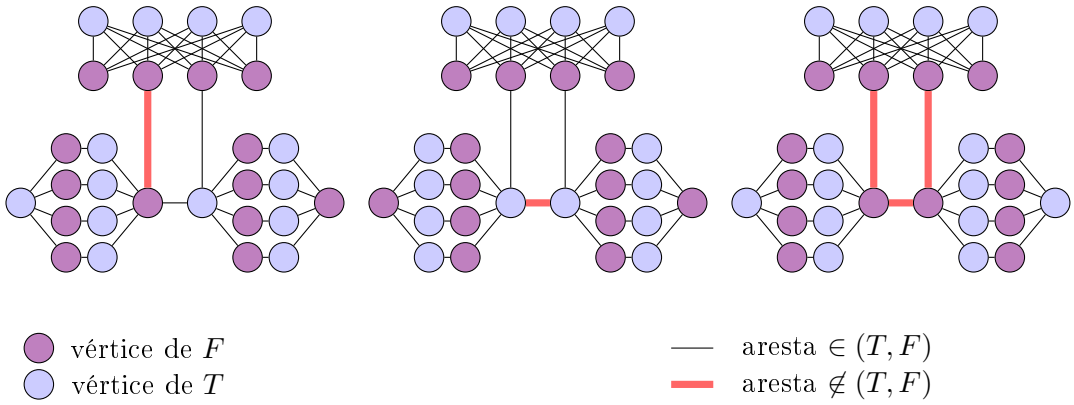
Temos que, em uma cláusula $C_i = (a_i, b_i)$, ou apenas uma das literais é satisfeita, ambas são ou nenhuma delas é.

1. Se apenas uma das literais é satisfeita, temos que, das arestas formadas pela cláusula C_i, duas estão no corte (T, F) e uma está fora.

2. Se as duas literais são satisfeitas, também teremos duas arestas no corte (T, F) e uma fora.

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

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



Sabe-se que temos $2k$ ou mais arestas de E_1 que estão no corte (T, F), pois, para cada uma das k cláusulas verdadeiras, teremos duas arestas de E_1 no corte, como o mostrado anteriormente.

Se houvesse alguma aresta duplicada (duas arestas que ligam os mesmos vértices), teríamos um problema na contagem de arestas dentro e fora do corte. Isso poderia ocorrer no caso de duas cláusulas serem idênticas, $C_i = C_j = (a_i, b_i)$. Isso faria com que houvesse uma aresta duplicada entre os vértices a_1 e b_i, mas isso não acontece, dado que no enunciado é assumido que as cláusulas são distintas. Também não há o problema de dois vértices estarem em cláusulas diferentes, dado que para cada cláusula, os vértices se ligam a um F_i diferente. E mesmo se repetirmos as variáveis na mesma cláusula, cada elemento da cláusula se ligará a um F_i diferente.

Mostramos que existe uma solução, em Simple Max Cut, para todas as entradas do problema MaxSat2 reduzidas para Simple Max Cut, de forma que a solução do Simple Max Cut leva a uma solução de Max Sat2. Agora mostraremos que qualquer solução obtida em uma redução para Simple Max Cut levará a uma solução de Max Sat2.

10.2 Redução SimpleMaxCut para Bissecção

11 Algoritmo de Jansen 11

12 Geração de árvores binárias aleatórias

Nesta seção falaremos sobre o gerador de árvores binárias aleatórias, um dos tipos de árvores que foi utilizado nos testes do algoritmo FST.

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

O gerador recebe um inteiro n e cria um vetor permutado aleatoriamente com valores de 0 a $n-1$. Depois, constrói uma árvore binária de busca e vai inserindo os nós de acordo com a ordem dada pelo vetor. Isso leva tempo $O(n^2)$, dado que a inserção na árvore binária de busca leva tempo linear para cada um dos n nós.

Segue abaixo o algoritmo do gerador.

Algorithm 6: Inserção do nó n na árvore binária de busca

```
   input  : raiz r da árvore e nó n
   output:
 1 if n.chave > r.chave then
 2 |   if r.right = NULL then
 3 |   |   r.right ← n;
 4 |   else
 5 |   |   Algoritmo 6(r.right, n);
 6 |   end
 7 else
 8 |   if r.left = NULL then
 9 |   |   r.left ← n;
10 |   else
11 |   |   Algoritmo 6(r.left, n);
12 |   end
13 end
```

Algorithm 7: Gerador de árvores binárias aleatórias

input : inteiro n
output: raiz da árvore binária aleatória gerada

```
   // Aleatoriza vetor
1  for i = 0 → n − 1 do
2  |  v[i] ← i;
3  end
4  for i = n − 1 → 1 do
5  |  indice ← RANDOM(0 . . . i);
6  |  v[indice] ↔ v[i];
7  end

   // Inserção dos elementos do vetor na árvore binária de busca
8  raiz.chave ← v[0];
9  for i = 1 → n − 1 do
10 |  new no.chave ← v[i];
11 |  Algoritmo 6(raiz, no);
12 end
13 return raiz;
```

Nota-se que numa árvore desse tipo, em grande parte dos casos, a maioria dos nós estará presente no caminho mais longo. O que favorece um pouco o algoritmo FST.

13 Experimentos computacionais 13

13.1 Árvores Binárias Aleatórias

13.2 Árvores Balanceadas

13.3 Árvores de Caminho Longo

14 Conclusões 14

Referências

[1] 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.

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

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

[4] Klaus Jansen, Marek Karpinsk, Andrzej Lingas, and Eike Seide. 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.

[5] Alexander Kampmeier. On the computation of local and global optima for soft power diagrams. master thesis, 2014.

[6] 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.

[7] Tina J. Schmidt. The minimum bisection problem. PhD Thesis in preparation.