3
Algoritmos para o PBH

Neste capítulo apresentamos dois algoritmos que propomos para o PBH. Um deles é (14/3)-aproximado, descrito a seguir. O outro é polinomial exato, descrito na Seção 3.2.

3.1
Algoritmo APPROX

Introduzimos nesta seção o algoritmo APPROX, um algoritmo (14/3)-aproximado para o problema PBH. Este algoritmo executa em tempo $O(n \log m)$ e em espaço linear. APPROX é similar ao algoritmo ASSIGN-HOTLINK, exceto por duas diferenças. A primeira é que $w(v, T)$ representa o número de hotleaves da sub-árvore binária de T, com raiz em v, que contêm o maior número de hotleaves. A segunda diferença é o fato da recursão parar apenas quando T tem exatamente uma hotleaf, o que permite remover as linhas 4 e 5 do pseudo-código do procedimento LOG na Figura 2.1, resultando no procedimento APPROX-LOG da Figura 3.6.

Antes de atribuir os hotlinks na árvore de entrada T, inicializamos o vetor $\mathbf{w}$. Isto é feito pelo procedimento INIT-W. Fornecemos o pseudo-código de INIT-W na Figura 3.6. A Figura 3.1.(a) dá um exemplo ilustrativo dos valores de $\mathbf{w}$ calculados em cada nó. Quando um hotlink é adicionado, alguns valores de $\mathbf{w}$ podem mudar, como ilustra a Figura 3.1.(b).

A corretude de APPROX segue um argumento similar ao utilizado na prova de corretude de ASSIGN-HOTLINK. Precisamos mostrar que sempre é possível encontrar o nó u na linha 6 do pseudo-código de APPROX-LOG. Para isso utilizamos o Lema 3.1 abaixo, semelhante ao Lema 2.1. De agora em diante assumimos que $t \in (0.5, 1)$.

Lema 3.1 *Seja T' uma sub-árvore de T com raiz em r', tendo $w(r', T') \geq 2$. Então, existe um nó $u \in T'$ tal que $w(u, T') \geq tw(r, T')$, e $w(v, T') < tw(r, T')$ para todo $v \in S(u, T')$.*

PUC-Rio - Certificação Digital Nº 0210481/CA

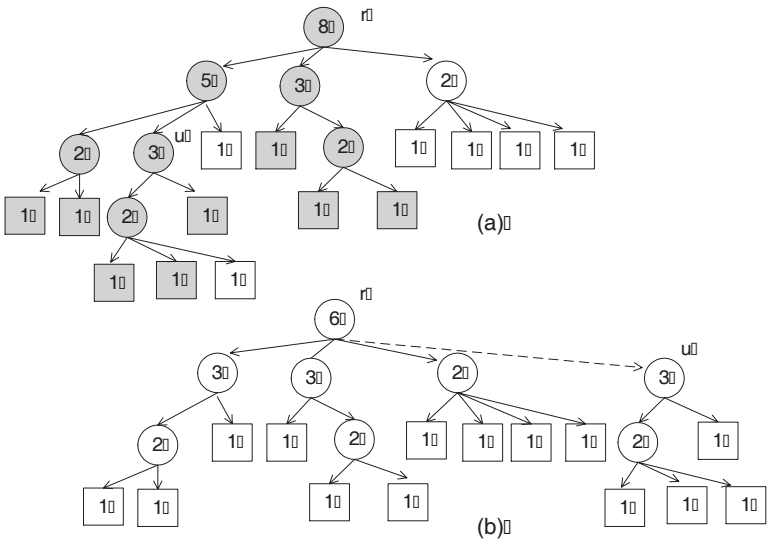

Figura 3.1: (a) e (b) mostram os valores de **w** para as árvores T e $T^{\{(r,u)\}}$, respectivamente. Os nós sombreados na Figura 3.1.(a) são os nós da sub-árvore binária, com raiz em r, com o maior número de hotleaves.

Prova. Podemos utilizar a prova feita para o Lema 2.1, pois continua sendo verdade que $w(b,T') \leq w(a,T')$ sempre que $b \in S(a,T')$. Além disso, sabemos que u não é folha, já que $t > 1/2$ e $w(r',T') \geq 2$. □

3.1.1 Fator de aproximação

Analizaremos agora o fator de aproximação de APPROX. Como temos o limite inferior do Corolário 3.2, provamos que APPROX tem um fator de aproximação constante mostrando que $h(T^A) = O(\log w(r,T))$, onde A é o conjunto de hotlinks construído por APPROX. Como no algoritmo ASSIGN-HOTLINK, a idéia chave é mostrar que gastamos no máximo dois links para obter sub-problemas cujo valor de **w** é reduzido por um fator constante. Utilizamos o Lema 3.3 para este propósito.

Corolário 3.2 $h^*(T) \geq \log_3 w(r,T)$.

Prova. Decorre do Teorema 2.7 e da definição de **w**. □

Lema 3.3 *Seja T' uma sub-árvore de T com raiz em r'. Seja u o nó selecionado na linha 6 quando APPROX-LOG é executado com entrada T' . Então,*

(i) $w(v,T'_v) < tw(r',T'), \forall v \in S(u,T')$;

(ii) $w(v,T'_v) \leq (1-t)w(r',T'), \forall v \in S(r',T') - S(u,T') - \{s_1\}$;

PUC-Rio - Certificação Digital Nº 0210481/CA

(iii) $w(s_1, T'_{s_1} - T'_u) \leq 2(1-t)w(r', T')$.

Prova. Para simplificar as equações, denotamos nesta prova $w(a, T')$ como $w(a)$, $\forall a \in T'$.

Pela escolha de u sabemos que $w(v) < tw(r'), \forall v \in S(u, T')$. Portanto, (i) decorre do fato de que $w(v) = w(v, T'_v)$.

Provamos agora (ii). Se $u = r'$, então $S(r', T') - S(u, T') = \emptyset$. Se $u \neq r'$, então $w(s_1) \geq w(u) \geq tw(r')$. Então, para cada $v \in S(r', T') - S(u, T') - \{s_1\}$, temos que $w(r') \geq w(s_1) + w(v)$. Logo, $w(v, T'_v) = w(v) \leq (1-t)w(r')$.

Finalmente, vamos demonstrar (iii). Seja s_i o ancestral do nó u no nível i em T', para $i = 1, \ldots, h$, onde h é o nível de u. Por conveniência, seja $s_0 = r'$. Usamos s'_i para denotar o irmão de s_i em T' com maior valor em $\mathbf{w}$, para $i = 1, 2, \ldots, h$. Se um nó s_i não tem um irmão, definimos $w(s'_i) = 0$. A Figura 3.2 mostra os nós s_i e s'_i no caminho que conecta u à raiz de T'.

PUC-Rio - Certificação Digital Nº 0210481/CA

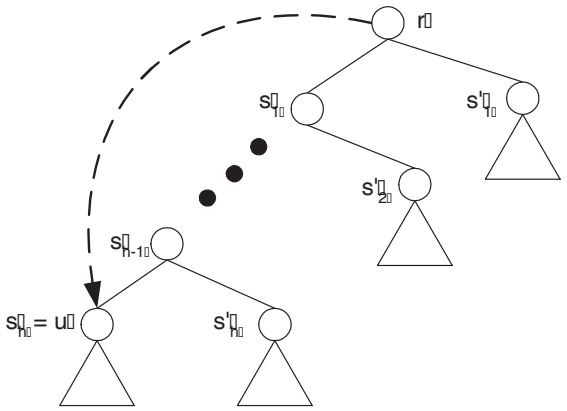

Figura 3.2: Os nós s_i e s'_i no caminho que conecta u à raiz de T'.

Pela escolha de $u = s_h$, temos que $w(s_h) \geq tw(r')$. Como resultado, temos também que $w(s_i) \geq tw(r')$, para $i = 1, 2, \ldots, h-1$. Como $t > 0.5$, temos que s_i maximiza $\mathbf{w}$ dentre todos os filhos de s_{i-1} em T'. Logo, temos que $w(s_i) = w(s_{i+1}) + w(s'_{i+1})$, para $i = 0, 1, \ldots, h-1$.

Pela Figura 3.2, fica claro que

$$w(s_1) = \sum_{i=2}^{h} w(s'_i) + w(u).$$

Como $w(s_1) \leq w(r')$ e $w(u) \geq tw(r')$, temos que

$$\sum_{i=2}^{h} w(s'_i) \leq (1-t)w(r') \tag{3-1}$$

Seja agora k o menor índice em $\{2, 3, \ldots, h\}$ tal que $w(s'_k, T'_{s_1} - T'_u) > w(s_k, T'_{s_1} - T'_u)$. Se este k não existe, fazemos $k = h$. Neste caso, temos que

$$w(s_1, T'_{s_1} - T'_u) \leq w(s'_k, T'_{s_1} - T'_u) + \sum_{i=2}^{k} w(s'_i, T'_{s_1} - T'_u).$$

Como a adição de (r', u) não modifica as sub-árvores com raiz em s'_i, para $i = 1, \ldots, h$, temos que

$$w(s_1, T'_{s_1} - T'_u) \leq w(s'_k) + \sum_{i=2}^{k} w(s'_i).$$

Como $k \leq h$, decorre da equação anterior e da Equação (3-1) que

$$w(s_1, T'_{s_1} - T'_u) \leq 2(1-t)w(r')$$

□

O lema a seguir fornece um limite superior para o custo $h()$ da árvore melhorada obtida por APPROX.

Lema 3.4 *Seja* $h_{APPROX}(T) = h(T^A)$, *onde* A *é a atribuição de hotlinks obtida no final da execução de APPROX. Então,* $h_{APPROX}(T) \leq c \log w(r, T) + 1$, *onde*

$$c = \max \left\{ \frac{2}{\log\left(\frac{1}{t}\right)}, \frac{1}{\log\left(\frac{1}{2(1-t)}\right)} \right\}.$$

Prova. A prova é similar a do Teorema 2.3, exceto por duas diferenças. O único caso base utilizado é para $w(r, T) = 1$. Além disso, aplicamos os limites superiores dados pelo Lema 3.3, no lugar daqueles dados pelo Lema 2.2, para fornecer limites superiores para lado direito da equação (2-1). □

Novamente, utilizamos cálculo para obter o valor ótimo de $t = \frac{9-\sqrt{17}}{8}$ (observe que $t > 0.5$). Neste caso, temos que $c = \frac{2}{3-\log(9-\sqrt{17})} < 2.801$, que conduz ao teorema a seguir.

Teorema 3.5 *O algoritmo APPROX é um algoritmo* (14/3)*-aproximado para o PBH.*

Prova. Se $w(r, T) = 1$, então APPROX sempre constrói uma atribuição de hotlinks ótima. Assim, vamos assumir que $w(r, T) > 1$. Segue do Corolário (3.2) e do Lema 3.4 que

PUC-Rio - Certificação Digital Nº 0210481/CA

$$\frac{h_{APPROX}(T)}{h^*(T)} \leq \frac{c \log w(r,T) + 1}{\log_3 w(r,T)}. \quad (3\text{-}2)$$

Porém, como a função objetivo do problema PBH é sempre inteira, temos também que

$$\frac{h_{APPROX}(T)}{h^*(T)} \leq \frac{\lfloor c \log w(r,T) + 1 \rfloor}{\lceil \log_3 w(r,T) \rceil}. \quad (3\text{-}3)$$

Estabelecemos este teorema usando (3-3) sempre que $w(r,T) \leq 125$, e (3-2) caso contrário. No primeiro caso, pode ser verificado que este teorema vale para todos os valores possíveis de $w(r,T)$. No segundo caso, temos que

$$\frac{h_{APPROX}(T)}{h^*(T)} \leq \frac{c \log w(r,T)}{\log_3 w(r,T)} + \frac{1}{\log_3 126} < 14/3.$$

□

3.1.2 Análise de complexidade

Analisaremos agora as complexidades de tempo e espaço do algoritmo APPROX.

Teorema 3.6 *O algoritmo APPROX executa em tempo $O(n \log m)$.*

Prova. Como $w(r,T) \leq |L| = m$, é suficiente mostrar que APPROX executa em tempo $O(n \log w(r,T))$.

O algoritmo INIT-W executa em tempo linear. Portanto, o tempo de execução $t(T)$ de APPROX, para uma entrada T, é dado por

$$t(T) = c_1 n + t(T_{s_1} - T_u) + \sum_{v \in S(u,T) \cup S(r,T) - \{s_1\}} t(T_v),$$

onde c_1 é uma constante positiva. Note que n,u,s_1 e r dependem do argumento T da função $t()$.

Vamos assumir indutivamente que $t(T') \leq c_1 |T'| \log w(r,T')$ para toda sub-árvore máxima T' com raiz em r', onde $w(r',T') < w(r,T)$. Usando as relações entre $w(r,T)$ e $w(v,T_v)$ dadas pelo Lema 3.3, pode-se concluir que $t(T) = O(n \log w(r,T)) = O(n \log m)$. □

Teorema 3.7 *O algoritmo APPROX executa em espaço linear.*

Prova. Como calculamos os valores de **w** para cada sub-árvore de entrada do algoritmo APPROX-LOG, podemos descartar os valores armazenados para as sub-árvores consideradas anteriormente. Deste modo, estaremos alocando no máximo n valores no vetor **w**. □

PUC-Rio - Certificação Digital Nº 0210481/CA

3.2 Algoritmo PATH

Nesta seção introduzimos o algoritmo PATH, um algoritmo exato baseado em programação dinâmica para resolver problemas de atribuição de hotlinks. Descrevemos neste seção como PATH fornece uma solução para o PBH. No capítulo 4 discutimos o algoritmo M-PATH, uma adaptação do algoritmo PATH para resolver o MBH.

Dado um parâmetro D definido pelo usuário, PATH encontra a melhor atribuição de hotlinks tal que a altura da árvore melhorada é no máximo D. Ou seja, uma atribuição de hotlinks A^*_D com as seguintes propriedades:

(i) $H(T^{A^*_D}) \leq D$;

(ii) para toda atribuição de hotlinks viável A tal que $H(T^A) \leq D$, temos que $h(T^{A^*_D}) \leq h(T^A)$.

Assim, nosso algoritmo resolve um *problema de atribuição de hotlinks com restrição de altura*. Como veremos, PATH executa em tempo $O(n3^D)$ e requer $O(n2^D)$ de espaço. Lembre que o Teorema 2.5 garante que existe uma árvore melhorada T^{A^*} ótima para o PBH, cuja altura é logarítmica em n. Portanto, se fixamos D como o limite de altura fornecido pelo Teorema 2.5, temos um algoritmo polinomial exato para o PBH. Por outro lado, não conhecemos limite de altura logarítmico para a árvore melhorada ótima no caso do MBH. Neste caso, o usuário pode controlar o parâmetro D para reduzir o tempo de execução, em detrimento da qualidade da solução.

Uma maneira direta de resolver o PBH usando programação dinâmica seria a seguinte. Para cada atribuição possível (r, u) do hotlinks da raiz r de T, obtenha a árvore melhorada $T^{\{(r,u)\}}$. Então, para cada filho v de r em $T^{\{(r,u)\}}$, resolva recursivamente o sub-problema onde a árvore de entrada é a sub-árvore máxima de $T^{\{(r,u)\}}$ com raiz em v. No final, retorne a melhor solução encontrada. Esta proposta, entretanto, gera um número exponencial em n de sub-problemas.

O modo como PATH decompõe um problema permite gerar um número exponencial em D de sub-problemas. A idéia chave é adiar a decisão de qual nó receberá o hotlink. Assim, PATH decide apenas qual sub-árvore filha receberá o hotlink. A Figura 3.3 ilustra como fazemos esta decomposição para o hotlink da raiz. Representamos com um nó preenchido quando o hotlink deste nó não está mais disponível. Portanto, temos na Figura 3.3.(b) os sub-problemas gerados pela decisão de apontar o hotlink de r para a sub-árvore filha mais a direita. Neste caso, o hotlink de r não

PUC-Rio - Certificação Digital Nº 0210481/CA

está disponível no sub-problema com as outras sub-árvores filhas. Por outro lado, vemos na Figura 3.3.(c) os sub-problemas gerados pela escolha de não utilizar o hotlink de r na sub-árvore filha mais a direita. Se c_1 e c_2 são os custos ótimos de cada sub-problema de um par de sub-problemas, então o custo do par será $\max\{c_1, c_2\}$. PATH tomará a decisão que gera o par de sub-problemas com menor custo.

Note que sempre que geramos um sub-problema avaliando a sub-árvore filha mais a direita, aumentamos o número de nós que precisamos decidir para onde apontar o hotlink. Por exemplo, observe que no sub-problema da direita na Figura 3.3.(b) temos que decidir também para onde apontar o hotlink de f. Com isso, a decomposição vai gerando um caminho direcionado que aponta para a raiz da sub-árvore que estamos considerando, como ilustra a Figura 3.5.(a). Alguns hotlinks no caminho podem não estar disponíveis. Assim, PATH deve testar todas as combinações, que é uma operação exponencial no comprimento do caminho. Mas não precisamos considerar caminhos maiores que a altura máxima da árvore melhorada, que estamos limitando pelo valor de D.

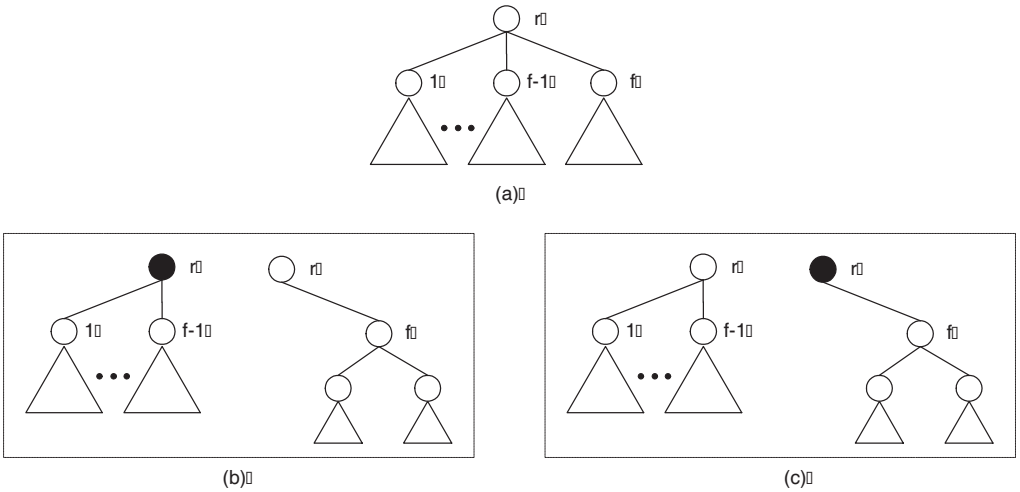

Figura 3.3: (a) árvore com raiz em r e f filhos. (b) par de sub-problemas gerado pela escolha de apontar o hotlink de r para a sub-árvore filha f. (c) par de sub-problemas gerado pela escolha de apontar o hotlink de r para alguma outra sub-árvore filha.

Agora que comentamos sobre o caminho gerado, podemos falar de mais um critério de decomposição. Devemos considerar também os sub-problemas em que um hotlink disponível no caminho aponta para a raiz, como ilustra a Figura 3.4. Deste modo, estaremos explorando todas as maneiras de utilizar os hotlinks. Além disso, é nesta etapa de decomposição que de fato atribuímos os hotlinks. Alertamos que em alguns sub-problemas

PUC-Rio - Certificação Digital Nº 0210481/CA

não podemos apontar um hotlink do caminho para a raiz, pois devemos obter uma atribuição de hotlinks viável. Esta situação ficará clara adiante, quando estivermos detalhando o algoritmo PATH.

Como podemos observar, esta estratégia de decomposição produz um problema mais geral que o PBH, que chamamos de P-PBH. Definimos o P-PBH a seguir.

Entrada:

i) um caminho direcionado $\mathbf{q} = (V_{\mathbf{q}}, E_{\mathbf{q}})$ onde $V_{\mathbf{q}} = \{q_1, \ldots, q_k\}$ e $E_{\mathbf{q}} = \{(q_i, q_{i+1}) | 1 \leq i \leq k-1\}$;

ii) um vetor $\mathbf{a} = (a_1, ..., a_k, a_{k+1}, b) \in \{0,1\}^{k+2}$;

iii) uma árvore $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ com raiz em r;

iv) um inteiro D.

Saída: Uma atribuição de hotlinks A na árvore $T_{\mathbf{q}} = (V_{\mathbf{q}} \cup \mathcal{V},\ E_{\mathbf{q}} \cup \mathcal{E} \cup \{(q_k, r)\})$, satisfazendo as seguintes seis condições:

(a) A é viável;

(b) nenhum hotlink pode apontar para um nó em $V_{\mathbf{q}}$;

(c) se $a_i = 0$, então nenhum hotlink pode partir de q_i;

(d) se $a_{k+1} = 0$, então nenhum hotlink pode partir de r;

(e) se $b = 0$, então nenhum hotlink pode apontar para r;

(f) $H(T^A_{\mathbf{q}}) \leq D$.

Objetivo: Minimizar o nível máximo de uma hotleaf na árvore melhorada resultante $T^A_{\mathbf{q}}$.

Observe que o PBH é um caso particular do P-PBH quando $\mathbf{q}$ é vazio, $b = a_1 = 1$, $D = H$ and $\mathcal{T} = T$. Assim, um algoritmo exato para o P-PBH é também um algoritmo exato para o PBH.

PUC-Rio - Certificação Digital Nº 0210481/CA

3.2.1
Resolvendo o P-PBH

As Figuras 3.4 e 3.5 são utilizadas nesta seção para ilustrar algumas idéias. As Figuras 3.4.(a) e 3.5.(a) representam uma instância do P-PBH onde o caminho $\mathbf{q}$ consiste de quatro nós q_1, q_2, q_3 e q_4. Se um nó $q_i \in \mathbf{q}$ é tal que $a_i = 1$, então q_i é dito ser *disponível*. O nó q_3 é o único nó não disponível, que representamos com um nó preenchido. A sub-árvore $\mathcal{T}$ tem raiz no nó r. Como $a_5 = 1$, o hotlink da raiz esta disponível. Além disso, como $b = 1$, os hotlinks podem ser apontados para r.

Precisamos introduzir algumas convenções: dados dois vetores binários $\mathbf{c}$ e $\mathbf{d}$, usamos $\mathbf{cd}$ para indicar o vetor obtido da concatenação de $\mathbf{c}$ e $\mathbf{d}$. Por exemplo, se $\mathbf{c} = (0,1)$ e $\mathbf{d} = (1,0,0)$, então $\mathbf{cd} = (0,1,1,0,0)$. Usamos $\mathbf{c}_i$ para denotar o vetor truncado no i-ésimo elemento de $\mathbf{c}$. No exemplo anterior, $\mathbf{c}_1 = (0)$ e $\mathbf{c}_2 = (0,1)$. Para um caminho direcionado $\mathbf{q}$ e um nó u, usamos $\mathbf{q} \rightarrow u$ para indicar o caminho obtido inserindo o nó u no final de $\mathbf{q}$. Usamos $\mathbf{q}_i$ para denotar o sub-caminho de $\mathbf{q}$ formado pelos i primeiros nós.

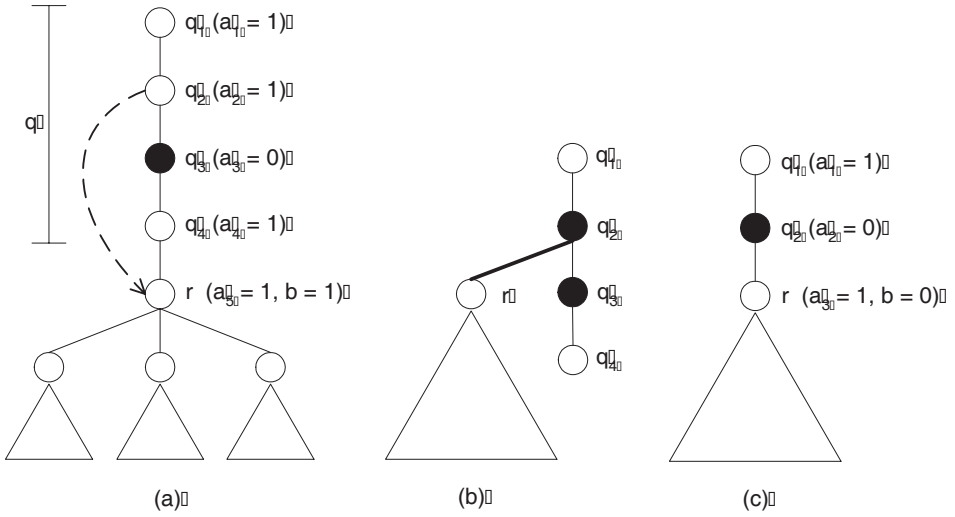

Figura 3.4: (a) uma instância do problema P-PBH. (b) a árvore melhorada obtida pela adição do hotlink (q_2, r). (c) o sub-problema correspondente gerado no caso 1.

Seja $h^*(\mathbf{q}, \mathbf{a}, \mathcal{T})$ o custo de uma solução ótima de uma instância do P-PBH definida por um vetor binário $\mathbf{a}$, uma árvore direcionada $\mathcal{T}$, um caminho $\mathbf{q}$ com $|\mathbf{a}| - 2$ nós, e um inteiro D. Se $|\mathbf{q}| > D$, PATH faz $h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = \infty$. Assim, vamos assumir que $|\mathbf{q}| \leq D$. De modo a resolver esta instância, devemos considerar os seguintes casos:

1. algum hotlink é atribuído de um nó em $\mathbf{q}$ para r na solução ótima;

PUC-Rio - Certificação Digital Nº 0210481/CA

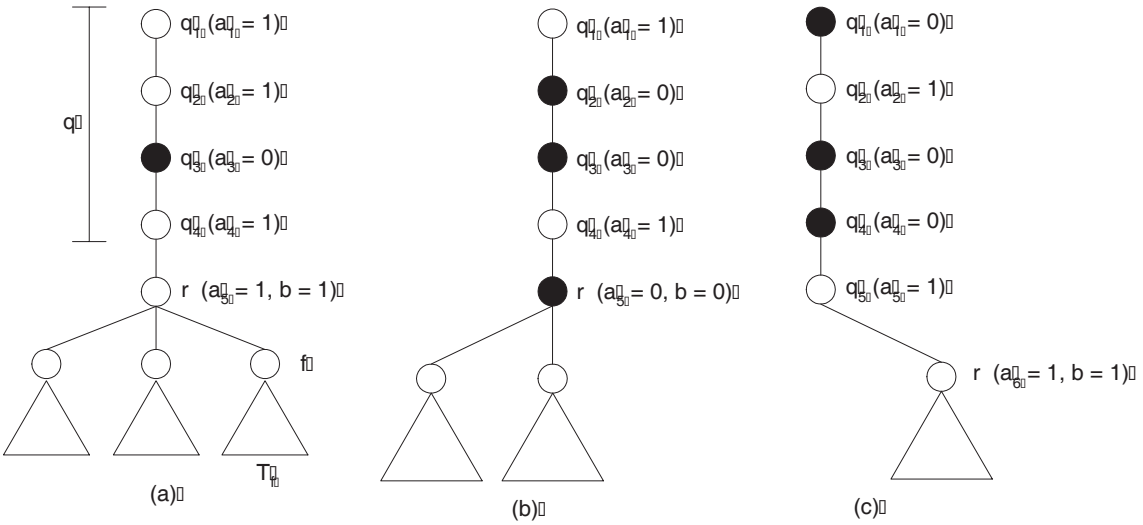

Figura 3.5: (a) uma instância do problema P-PBH. (b) e (c) a decomposição no caso 2, quando $c = (0, 1, 0, 0, 1)$.

2. nenhum hotlink é atribuído de um nó em $\mathbf{q}$ para r na solução ótima.

Caso 1 Este caso é considerado apenas quando $b = 1$. Neste caso, devemos adicionar um hotlink de algum nó disponível para r. Assim, temos $\sum_{i=1}^{k} a_i$ possibilidades. Por exemplo, se (q_2, r) é atribuído na árvore da Figura 3.4.(a), então obtemos a árvore da Figura 3.4.(b). Porém, como q_3 e q_4 não são ancestrais de hotleaves, eles podem ser removidos sem modificar o custo da solução. Deste modo, obtemos o sub-problema da Figura 3.4.(c). Observe que b recebe valor 0, pois a condição (ii) da definição de atribuição de hotlinks viável garante que nenhum nó pode receber dois hotlinks. Em geral, se algum hotlink aponta para r na solução ótima, temos que

$$h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = \min_{i \in \{1,2,3,\ldots,k\} \mid a_i = 1} \{h^*(\mathbf{q}_i, \mathbf{a}_{i-1}(0, a_{k+1}, 0), \mathcal{T})\} \qquad (3\text{-}4)$$

Caso 2 Neste caso, todos os nós disponíveis em $\mathbf{q}$ podem apontar apenas para algum nó em $\mathcal{V} - \{r\}$. Assim, PATH deve decidir quais nós disponíveis podem apontar para nós em $\mathcal{T}_f$, a sub-árvore máxima com raiz no último filho f de r (assumindo qualquer ordem). Seja $k' = \sum_{i=1}^{k+1} a_i$ o número de nós disponíveis. Então, PATH tem $2^{k'}$ possibilidades para tomar esta decisão. Como não é claro qual delas é a melhor, todas devem ser consideradas.

Para ilustrar esta idéia, vamos considerar o sub-problema da Figura 3.5.(a) e a possibilidade onde q_2 e r permanecem disponíveis para $\mathcal{T}_f$ (vide Figura 3.5.(c)). Como consequência, apenas q_1 e q_4 poderão apontar para nós

PUC-Rio - Certificação Digital Nº 0210481/CA

em $\mathcal{T} - \mathcal{T}_f$ (Figura 3.5.(b)). Portanto, a Figura 3.5.(b) define um novo sub-problema ($\mathbf{q}$,$\mathbf{a}'$,$\mathcal{T} - \mathcal{T}_f$), onde $\mathbf{a}' = (1, 0, 0, 1, 0, 0)$. Note que b recebe valor 0 pois estamos no caso 2 (hotlinks do caminho não apontam para a raiz). Por outro lado, a Figura 3.5.(c) define um novo sub-problema ($\mathbf{q} \rightarrow r$,$\mathbf{a}''$,$\mathcal{T}_f$), onde $\mathbf{a}'' = (0, 1, 0, 0, 1, 1, 1)$.

Logo, o máximo entre os custos das soluções ótimas dos sub-problemas definidos pelas Figuras 3.5.(b) e 3.5.(c) é o custo da solução ótima para o problema da Figura 3.5.(a), considerando que nenhum hotlink pode ser atribuído para r (caso 2), os nós q_2 e r não podem apontar para nós em $\mathcal{T} - \mathcal{T}_f$, e os nós q_1 e q_4 não podem apontar para nós em $\mathcal{T}_f$.

Em geral, seja C o conjunto de vetores binários definido por $C = \{(c_1, \ldots, c_{k+1}) \mid c_i \leq a_i \text{ para } i = 1, \ldots, k+1\}$. Cada $\mathbf{c} \in C$ corresponde a uma das $2^{k'}$ possibilidades de selecionar os nós que permanecerão disponíveis para apontar para nós em $\mathcal{T}_f$. Além disso, seja $\overline{\mathbf{c}} = \mathbf{a} - \mathbf{c}$. Este vetor define quais nós de $\mathbf{q}$ permanecerão disponíveis para apontar para nós em $\mathcal{T} - \mathcal{T}_f$. Então, considerando todas as escolhas para $\mathbf{c}$, temos que

$$h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = \min_{\mathbf{c} \in C} \max \left\{ \begin{array}{l} h^*(\mathbf{q} \rightarrow r, \mathbf{c}(1,1), \mathcal{T}_f), \\ h^*(\mathbf{q}, \overline{\mathbf{c}}(0), \mathcal{T} - \mathcal{T}_f) \end{array} \right\} \tag{3-5}$$

Casos 1 e 2 juntos: Sejam P e Q o lado direito das equações (3-4) e (3-5), respectivamente. Assim,

$$h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = \left\{ \begin{array}{l} \text{Q, se } b = 0 \\ \min\{P, Q\}, \text{ se } b = 1 \end{array} \right.$$

Condição de parada: Se $\mathcal{T}$ tem apenas um nó e este nó é uma hotleaf l, então a melhor escolha é atribuir um hotlink do primeiro nó disponível em $\mathbf{q}$ para l. Logo,

$$h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = \left\{ \begin{array}{l} \min\{i | 1 \leq i \leq k \text{ e } a_i = 1\} \text{ se } \mathbf{q} \text{ tem hotlink disponível} \\ k, \text{ caso contrário} \end{array} \right. \tag{3-6}$$

Se $\mathcal{T}$ é vazio, então $h^*(\mathbf{q}, \mathbf{a}, \mathcal{T}) = 0$.

3.2.2 Análise

Para provar que o algoritmo PATH é correto, devemos mostrar que toda atribuição de hotlinks viável A, com $H(T^A) \leq D$, é considerada

PUC-Rio - Certificação Digital Nº 0210481/CA

pelo algoritmo PATH. Suponha que os hotlinks do sub-problema $\mathbf{I}' = (\mathbf{q}', \mathbf{a}', T')$ foram considerados na construção da atribuição A. Portanto, $\mathbf{q}'$ é também um caminho em T^A. Como $H(T^A) \leq D$, temos que sub-problemas com $|\mathbf{q}'| > D$ podem ser descartados. De fato, sub-problemas com esta propriedade são os únicos descartados por PATH.

Agora, analizaremos as complexidades de tempo e de espaço do algoritmo PATH.

Teorema 3.8 *O algoritmo PATH executa em tempo $O(n3^D)$ e usa $O(n2^D)$ de espaço.*

Prova. Primeiro, vamos analisar o número de sub-problemas gerados. Como utilizamos a técnica de programação dinâmica, este número tem a mesma ordem do espaço utilizado pelo algoritmo PATH. Seja $(\mathbf{q}, \mathbf{a}, \mathcal{T})$ um sub-problema gerado quando PATH é executado para resolver o PBH. Note que $\mathbf{a}$ é um vetor binário com no máximo $D+2$ elementos. Além disso, $\mathcal{T}$ é uma sub-árvore de T removendo-se de T_v as sub-árvores com raiz nos últimos q filhos de v, onde $q \in \{0, 1, \ldots, |S(v,T)| - 1\}$ e v algum nó de T. Portanto, temos $O(\sum_{i=0}^{D} 2^{i+2}) = O(2^D)$ valores possíveis para $\mathbf{a}$ e $O(n)$ sub-árvores possíveis para $\mathcal{T}$, que fornece $O(n2^D)$ sub-problemas.

Para obter a complexidade de tempo do algoritmo PATH, contamos o número de sub-problemas verificados para calcular cada valor de $h^*(\mathbf{q}, \mathbf{a}, \mathcal{T})$. No caso 1, PATH verifica $O(D)$ sub-problemas, pois temos $O(D)$ maneiras possíveis de atribuir um hotlink de um nó em $\mathbf{q}$ para a raiz de $\mathcal{T}$. Por outro lado, no caso 2 o número de sub-problemas verificados por PATH depende do número de elementos diferentes de zero em $\mathbf{a}$. Se $\mathbf{a}$ tem j elementos não nulos, então $O(2^j)$ sub-problemas são verificados, pois esta é a complexidade do número de distribuições possíveis destes hotlinks entre os dois sub-problemas. Além disso, temos $O\left(n\binom{D+2}{j}\right)$ sub-problemas com j elementos não nulos, para $j = 0, 1, \ldots, D+2$. Logo, temos que a complexidade de tempo do algoritmo PATH é dada por

$$\begin{aligned} O(n2^D D) + \sum_{j=0}^{D+2} O\left(n\binom{D+2}{j}2^j\right) &= O(n2^D D) + O(n(2+1)^{D+2}) \\ &= O(n3^D). \end{aligned}$$

□

Corolário 3.9 *Para o problema PBH, o algoritmo PATH executa em tempo $O(n(nm)^{2.284})$ e usa $O(n(nm)^{1.441})$ de espaço.*

PUC-Rio - Certificação Digital Nº 0210481/CA

Prova. O Teorema 2.5 garante que existe uma solução ótima para o PBH com altura no máximo $1.441(\log m + \log n) + 2$. Então, PATH pode ser executado com $D = 1.441(\log m + \log n) + 2$, que implica nas complexidade acima. □

PUC-Rio - Certificação Digital Nº 0210481/CA

Algoritmo 3: INIT-W
Entrada: T: árvore direcionada
(1) **foreach** nó u em uma busca em profundidade em T
(2) **if** $u \in L$ **then** $w(u,T) \leftarrow 1$
(3) **if** u é uma folha em T **and** $u \notin L$ **then** $w(u,T) \leftarrow 0$
(4) **if** u tem apenas um filho a **then** $w(u,T) \leftarrow w(a,T)$
(5) Sejam a e b os filhos de u com os maiores valores em **w**
(6) $w(u,T) \leftarrow w(a,T) + w(b,T)$

Algoritmo 4: APPROX-LOG
Entrada: T': árvore direcionada
(1) $r' \leftarrow$ raiz de T'
(2) **if** T' tem exatamente uma hotleaf l
(3) $A \leftarrow A \cup \{(r', l)\}$
(4) **else**
(5) INIT-W(T')
(6) Encontre um nó u em T' tal que $w(u,T') \geq tw(r',T')$, e $w(v,T') < tw(r',T')$ para todo $v \in S(u,T')$.
(7) $A \leftarrow A \cup \{(r', u)\}$
(8) **if** $u \neq r'$ **then** seja s_1 o ancestral de u em $S(r',T')$
(9) **if** s_1 é definido **and** $s_1 \neq u$
(10) APPROX-LOG($T'_{s_1} - T'_u$)
(11) **foreach** $v \in S(u,T') \cup S(r',T') - \{s_1\}$
(12) APPROX-LOG(T'_v)

Algoritmo 5: APPROX
Entrada: T: árvore direcionada
Saída: A: atribuição de hotlinks
(1) Leia T; $L \leftarrow$ conjunto de folhas de T; $A \leftarrow \emptyset$
(2) APPROX-LOG(T)
(3) **return** A

Figura 3.6: Pseudo-código do algoritmo APPROX.

PUC-Rio - Certificação Digital Nº 0210481/CA