3
Busca com Custos de Acesso

Neste capítulo, apresentamos algoritmos aproximados para construção de estratégias de busca em vetores ordenados com custos de acesso variados.

3.1
Definições e Notação

Seja $\mathcal{A} = [a_1, \ldots, a_n]$ um vetor ordenado onde $c(a_i)$ é o custo de acessar o seu i-ésimo elemento. Definimos uma árvore binária de busca (ABB) T associada ao vetor $\mathcal{A}$ recursivamente da seguinte forma. T tem um nó raiz r associado a uma chave a_k de $\mathcal{A}$. Se $k > 1$, então r tem um nó filho à esquerda s_1, que é raiz de uma ABB associada ao subvetor $[a_1, \ldots, a_{k-1}]$. Em caso contrário, se $k = 1$, então r não tem nenhum filho à esquerda. Além disso, se $k < n$, então r tem um nó filho à direita s_2, que é raiz de uma ABB associada ao subvetor $[a_{k+1}, \ldots, a_n]$. Em caso contrário, se $k = n$, então r não tem nenhum filho à direita. Ao longo deste capítulo, podemos utilizar a mesma notação para denotar uma chave de $\mathcal{A}$ e o nó correspondente em uma ABB associada, indicando apenas se o objeto referido é uma chave ou um nó.

Qualquer estratégia de busca para $\mathcal{A}$ pode ser representada por uma ABB T associada a $\mathcal{A}$. Por exemplo, a Figura 3.1.(a) ilustra uma ABB T associada ao vetor ordenado $\mathcal{A} = [a_1, \ldots, a_8]$, representado na Figura 3.1.(b). Neste caso, cada nó de T aparece imediatamente acima da chave correspondente em $\mathcal{A}$. Neste caso, considere uma busca pelo valor da chave a_3 segundo a estratégia determinada por T. Primeiro, testamos a chave a_5, que corresponde à raiz de T. Como $a_3 < a_5$, passamos para o filho à esquerda de a_5 em T, que é a_2. Desta vez, temos $a_3 > a_2$. Por isto, passamos ao filho à direita de a_2, que é a_4. Em seguida, testamos a chave a_3, finalizando a busca. O custo desta busca é dado pela soma dos custos das chaves acessadas, que é $c(a_5) + c(a_2) + c(a_4) + c(a_3)$.

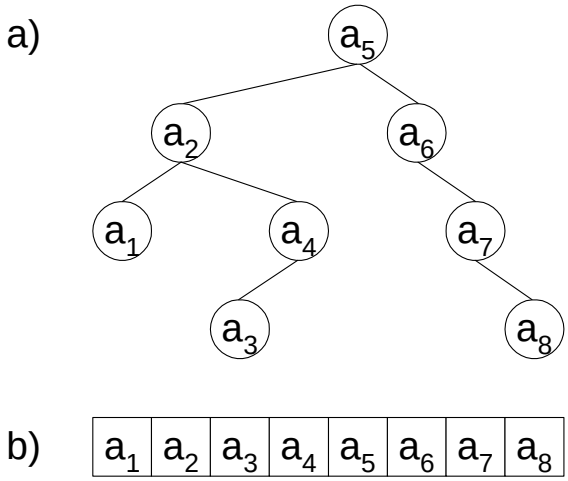

Figura 3.1: (a) Um exemplo de árvore binária de busca. (b) O vetor ordenado $\mathcal{A}$ associado.

Agora, apresentamos mais algumas definições utilizadas neste capítulo. Uma árvore estritamente binária de busca (AEBB) é uma ABB tal que todo nó interno tem exatamente dois filhos. O nível de um nó v em uma ABB T é dado pelo número de arestas do único caminho que liga v ao nó raiz de T.

Definimos o conceito de *ancestral* em uma árvore, recursivamente, da seguinte forma. Todo nó é ancestral dele próprio. Se s é um nó, então o pai de qualquer ancestral de s também é ancestral de s.

Dizemos que um nó s_1 é um *ancestral estrito* de um nó s_2 em uma ABB T quando s_1 é um ancestral de s_2 em T e $s_1 \neq s_2$. Além disto, dizemos que um nó s_1 é um *descendente* (*estrito*) de um nó s_2 quando s_2 é um ancestral (*estrito*) de s_1.

Dada uma ABB T, definimos recursivamente um percurso *em ordem* em T da seguinte forma. Se r é a raiz de T e T_1 e T_2 são as sub-árvores à esquerda e à direita, respectivamente, então este percurso corresponde a um percurso *em ordem* em T_1, uma visita a r e um percurso *em ordem* em T_2, nesta ordem.

Finalmente, utilizamos a expressão $\log x$ para denotar $\log_2 x$, pois a base dois para a função logaritmo é muito mais comum nesta tese do que as demais bases.

3.1.1 Custos

Pela discussão anterior, o PBC pode ser visto como um problema de construção de ABBs que minimizam uma certa função de custo. Agora,

definimos as duas funções de custo para ABBs consideradas neste capítulo.

Seja $N(a,T)$ o conjunto de todos os ancestrais do nó a na ABB T. Neste caso, o custo de buscar a chave a em T é dado por

$$\sum_{x \in N(a,T)} c(x).$$

Neste caso, o custo de T segundo o PBCM (Problema da Busca de Custo Médio) para o vetor de entrada $\mathcal{A}$ é dado por

$$EC(T) = \frac{1}{n} \sum_{i=1}^{n} \sum_{x \in N(a_i,T)} c(x).$$

Observe que esta função de custo assume que a probabilidade de buscar qualquer chave a_i é $1/n$. Para permitir expressar recursivamente a função de custo anterior de forma mais simples, descartamos o termo $1/n$ e reescrevemos esta função da seguinte forma:

$$c(T) = \sum_{i=1}^{n} \sum_{x \in N(a_i,T)} c(x). \tag{3-1}$$

Por outro lado, o custo de T segundo o PBPC (Problema da Busca de Pior Custo) para o vetor de entrada $\mathcal{A}$ é dado por

$$w(T) = \max_{i=1,\ldots,n} \left\{ \sum_{x \in N(a_i,T)} c(x) \right\}. \tag{3-2}$$

Com isto, dizemos que $c(T)$ e $w(T)$ são respectivamente o *custo médio de busca* e o *pior custo de busca* de T. Ao longo deste capítulo, assumimos sem perda de generalidade que $\sum_{i=1}^{n} c(a_i) = 1$.

Seja L o vetor correspondente a um dado subintervalo do vetor de entrada A. Utilizamos a seguinte notação ao longo deste capítulo:

$c_R(L)$**:** custo médio de busca da ABB construída pelo Algoritmo da Razão para o subvetor L;

$c(L)$**:** custo médio de busca da ABB construída pelo Algoritmo ECBM para o subvetor L;

$w(L)$**:** custo médio de busca da ABB construída pelo Algoritmo ECPB para o subvetor L;

PBCM(L)**:** instância do PBCM cujo vetor do entrada é o subvetor L;

PBPC(L): instância do PBPC cujo vetor do entrada é o subvetor L;

$c^*(L)$**:** custo médio de busca de uma solução ótima para o PBCM(L);

$w^*(L)$**:** pior custo de busca de uma solução ótima para o PBCM(L);

3.2 Busca com Custos Uniformes

Nesta seção, apresentamos propriedades das ABBs associadas a buscas binárias. Estas propriedades são utilizadas ao longo deste capítulo. Vale mencionar que a busca binária é uma estratégia ótima tanto para o PBCM quanto para o PBPC quando os custos de acesso são todos iguais a d. Com isto, temos a seguinte proposição.

Proposição 3.1 *Seja $L = [a_1, \ldots, a_n]$ um subvetor do vetor de entrada. Se temos $c(a_i) = d$ para $i = 1, \ldots, n$, então*

$$n(\log n - 1) \leq c^*(L)/d \leq n(\log n + 1).$$

Prova. Seja T a ABB que corresponde à busca binária para o subvetor L. Primeiro, observamos que T tem o número máximo de nós em cada nível menor do h, onde h é a altura de T. Como o número máximo de nós em nível i é 2^i, o número de nós em níveis menores do que h é sempre dado por

$$\sum_{i=0}^{h-1} 2^i = 2^h - 1.$$

Daí, concluímos que $2^h \leq n \leq 2^{h+1} - 1$, ou seja, que $h = \lfloor \log n \rfloor$. Além disso, obtemos que

$$c^*(L)/d = \sum_{i=0}^{h-1} (i+1)2^i + (h+1)(n - 2^h + 1). \tag{3-3}$$

Logo, temos

$$\begin{aligned} c^*(L)/d &\leq \sum_{i=1}^{h-1} (h+1)2^i + (h+1)(n - 2^h + 1) \\ &= (\lfloor \log n \rfloor + 1)n \leq n(\log n + 1). \end{aligned}$$

Portanto, falta provar apenas que $c^*(L)/d \geq n(\log n - 1)$. A partir de (3-3), temos

$$\begin{aligned} c^*(L)/d &= (h-1)2^h + 1 + (h+1)(n - 2^h + 1) \\ &= (h+1)n - 2^{h+1} + h + 2 \\ &> (\log n - (\log n - h) + 1)n - 2^{h+1} \\ &= (\log n - 1)n + (2 - (\log n - h) - 2^{h+1}/n)n \\ &= (\log n - 1)n + (2 - (\log n - h) - 2^{1-(\log n - h)})n. \end{aligned}$$

Definindo $x = 2^{1-(\log n - h)}$, podemos reescrever a desigualdade anterior como

$$c^*(L)/d > (\log n - 1)n + (1 + \log x - x)n.$$

Além disso, por definição de x e h, x pertence ao intervalo real $(1, 2]$. Logo, se $f(x) = 1 + \log x - x$, então basta provar que $f(x) \geq 0$ para qualquer $x \in (1, 2]$. Observe que $f(x)$ é contínua e derivável no intervalo $(1, 2]$. Sua derivada é dada por

$$f'(x) = \frac{1}{x \ln 2} - 1.$$

Como $f(1) = f(2) = 0$ e $f'(x)$ é sempre decrescente para $x \in (1, 2]$, concluímos que $f(x) \geq 0$ para qualquer $x \in (1, 2]$, o que conclui esta prova. □

No caso do PBPC, é fácil verificar que pior custo da busca binária para um dado subvetor L onde todas as chaves têm custo d é dado por

$$w^*(L) = d \times \lceil \log(n+1) \rceil.$$

Por isto, a prova da proposição apresentada a seguir é imediata.

Proposição 3.2 *Seja $L = [a_1, \ldots, a_n]$ um subvetor do vetor de entrada. Se temos $c(a_i) = d$ para $i = 1, \ldots, n$, então*

$$\log(n+1) \leq w^*(L)/d \leq \log(n+1) + 1.$$

3.3 Algoritmo da Razão

Nesta seção, apresentamos o Algoritmo da Razão para o PBCM. Este algoritmo roda em tempo $O(nh_R)$, onde h_R é a altura da ABB obtida. Como $h_R \leq n$, este tempo de execução é $O(n^2)$ no pior caso. Também provamos que

$$c_R(\mathcal{A}) \leq c^*(\mathcal{A}) \leq 4\ln(1+n), \tag{3-4}$$

para qualquer distribuição dos custos ao longo do vetor $\mathcal{A}$. Observe que o lado direito de (3-4) não está dividido por n devido à retirada do termo $1/n$ na definição dada por (3-1). Finalmente, apresentamos alguns experimentos comparativos para duas estruturas de custo específicas.

O Algoritmo da Razão constrói uma ABB através de uma abordagem *top-down*, utilizando uma regra simples para escolher a raiz da sub-árvore associada ao subvetor corrente. A Tabela 3.1 mostra uma pseudocódigo para o Algoritmo da Razão.

Algoritmo da Razão

raiz $\leftarrow$ Raiz$(1, n)$;

Função Raiz(i, m): nó;
  Se $i \leq m$ **então**
    $k^* \leftarrow i$;
    Para $k = i, \ldots, m$ **faça**
      Se $\frac{c_k}{\min\{k-i, m-k\}+1} \leq \frac{c_{k^*}}{\min\{k^*-i, m-k^*\}+1}$ **então**
        $k^* \leftarrow k$;
      Fim Se;
    Para;
    a_{k^*}.esquerda $\leftarrow$ Raiz(i, k^*-1);
    a_{k^*}.direita $\leftarrow$ Raiz(k^*+1, m);
    retornar a_{k^*};
  Senão
    retornar **nada**;
  Fim Se;
Fim Função.

Tabela 3.1: Um pseudocódigo para o Algoritmo da Razão.

Seja T_R a ABB obtida segundo o pseudocódigo da Tabela 3.1. Neste pseudocódigo, a função a Raiz é chamada recursivamente para construir cada sub-árvore de T_R. Ao final da construção, o nó raiz de T_R é guardado na

variável raiz. Para construir a sub-árvore associada ao subvetor $[a_i, \ldots, a_m]$, o algoritmo escolhe como nó raiz a chave a_k^* que minimiza

$$\frac{c_{k^*}}{\min\{k^* - i, m - k^*\} + 1}.$$

Em seguida, ele chama recursivamente a função Raiz para construir as sub-árvores associadas aos subvetores $[a_i, \ldots, a_{k^*-1}]$ e $[a_{k^*+1}, \ldots, a_m]$. As raízes das sub-árvores associadas a estes subvetores são conectadas ao nó a_k^* como filho à esquerda e como filho à direita, respectivamente. No pseudocódigo da Tabela 3.1, os atributos a_{k^*}.esquerda e a_{k^*}.direita denotam respectivamente os filhos à esquerda e à direita do nó a_{k^*}. Se o nó a_k^* não tem filho à esquerda ou à direita, então o atributo correspondente recebe o valor **nada**.

O critério de escolha adotado pelo Algoritmo da Razão para selecionar a raiz do subvetor corrente tenta balancear dois objetivos:

(i) dividir o subvetor em duas partes não muito diferentes;

(ii) selecionar uma chave de custo baixo.

Observe que este critério de escolha pode ser visto como uma seleção da chave de custo mínimo penalizado por um fator multiplicativo. Este fator aplica penalizações maiores aos custos das chaves mais próximas às extremidades do vetor. Por exemplo, de o subvetor atual é $[a_3, a_4, a_5, a_6, a_7, a_8]$, então a_3 e a_8 recebem uma mesma penalização $1 = \frac{1}{\min\{3-3,8-3\}+1} = \frac{1}{\min\{8-3,8-8\}+1}$, a_4 e a_7 recebem uma mesma penalização $1/2$, e a_4 e a_5 recebem uma mesma penalização $1/3$.

3.3.1
Análise da Solução

Agora, apresentamos uma análise da ABB construída pelo algoritmo da Razão. Para limitar o custo médio de busca desta árvore, utilizamos a seguinte proposição.

Proposição 3.3 *Se o Algoritmo da Razão seleciona a chave a_{k^*} como raiz da árvore construída para o subvetor $[a_1, \ldots, a_n]$, então $c_{k^*} \leq \frac{4C\min\{k^*, n-k^*+1\}}{n(n+2)}$, onde $C = \sum_{i=1}^{n} a_i$.*

Prova. Como a chave a_{k^*} foi selecionada, ela satisfaz às seguintes desigualdades:

$$c(a_i) \geq c(a_{k^*}) \frac{\min\{i, n-i+1\}}{\min\{k^*, n-k^*+1\}} \quad \text{para } i = 1, \ldots, n.$$

Somando as desigualdades anteriores, obtemos

$$\sum_{i=1}^{n} c(a_i) \geq c(a_{k^*}) \frac{\sum_{i=1}^{\lceil n/2 \rceil} i + \sum_{i=\lceil n/2 \rceil + 1}^{n} (n-i+1)}{\min\{k^*, n-k^*+1\}} \tag{3-5}$$

$$\geq c(a_{k^*}) \frac{n(n+2)}{4 \min\{k^*, n-k^*+1\}}, \tag{3-6}$$

pois temos

$$\sum_{i=1}^{n/2} i + \sum_{i=n/2+1}^{n} n-i+1 = \frac{n(n+2)}{4},$$

para valores pares de n, e

$$\sum_{i=1}^{(n+1)/2} i + \sum_{i=(n+1)/2+1}^{n} n-i+1 = \frac{(n+1)^2}{4} > \frac{n(n+2)}{4}.$$

para valores ímpares de n.

Como $\sum_{i=1}^{n} c(a_j) = C$, podemos deduzir a partir de (3-5) que

$$c(a_{k^*}) \leq \frac{4C \min\{k^*, n-k^*+1\}}{n(n+2)}.$$

□

Observe que a proposição anterior pode ser aplicada a qualquer subvetor do vetor de entrada $\mathcal{A}$. No próximo teorema, provamos por indução um limite superior para o custo médio de busca da árvore T_R.

Teorema 3.4 *O custo médio de busca da árvore T_R, construída para o vetor de entrada $[a_1, \ldots, a_n]$, não é maior que $4 \ln(n+1)$.*

Prova. Provamos que $c(T_R) \leq 4C \ln(n+1)$ por indução sobre o número de elementos n do vetor de entrada, onde $C = \sum_{i=1}^{n} c(a_j) = 1$. Para $n = 1$, o teorema vale pois $c(T_R) = c(a_1) = 1 < 4 \ln 2$. Agora, assumindo que o resultado vale para qualquer vetor cujo número de elementos é menor do que um dado n, basta provar que ele também vale para o vetor $[a_1, \ldots, a_n]$. Observe que esta hipótese indutiva também pode ser aplicada a qualquer subvetor do vetor $[a_1, \ldots, a_n]$.

Seja a_{k^*} a chave selecionada pelo Algoritmo da Razão como raiz da árvore T_R associada a este vetor. Além disso, assumimos sem perder generalidade que $k^* \leq \lceil n/2 \rceil$. Observe que os casos onde $k^* > \lceil n/2 \rceil$ são análogos por simetria. Com isto, o custo $c(T_R)$ da árvore T_R é dado por

$$c(T_R) = nc(a_{k^*}) + c(T(1, k^* - 1)) + c(T(k^* + 1, n)), \tag{3-7}$$

onde $T(1, k^* - 1)$ e $T(k^* + 1, n)$ são as sub-árvores construídas pelo Algoritmo da Razão para os subvetores $[a_1, \ldots, a_{k^*-1}]$ e $[a_{k^*+1}, \ldots, a_n]$, respectivamente. Sejam $C_1 = \sum_{i=1}^{k^*-1} c(a_i)$ e $C_2 = \sum_{i=k^*+1}^{n} c(a_i)$. Neste caso, concluímos a partir de (3-7) e da hipótese indutiva que

$$c(T_R) \leq nc(a_{k^*}) + 4C_1 \ln(k^*) + 4C_2 \ln(n - k^* + 1).$$

Como $\ln(k^*) \leq \ln(n - k^* + 1)$ para $1 \leq k^* \leq \lceil n/2 \rceil$ e $C_1 + C_2 < C$, concluímos que

$$\begin{aligned} c(T_R) &\leq nc(a_{k^*}) + 4(C_1 + C_2) \ln(n - k^* + 1) \\ &< nc(a_{k^*}) + 4C \ln(n - k^* + 1), \end{aligned}$$

Por outro lado, como $k^* \leq \lceil n/2 \rceil$, concluímos a partir da Proposição 3.3 que

$$c(a_{k^*}) \leq \frac{4Ck^*}{n(n+2)}.$$

Logo, temos

$$c(T_R) < \frac{4Ck^*}{n+2} + 4C \ln(n - k^* + 1).$$

Seja

$$f(k^*) = \frac{4Ck^*}{n+2} + 4C \ln(n - k^* + 1).$$

Derivando, obtemos

$$f'(k^*) = \frac{4C}{n+2} - \frac{4C}{n - k^* + 1}.$$

Como $f'(k^*)$ é negativa para $k^* \in [0, n]$, $f(k^*)$ atinge seu valor máximo dentro do intervalo $[0, \lceil n/2 \rceil]$ quando $k^* = 0$. Portanto, concluímos que

$$c(T_R) < f(0) = 4C\ln(n+1) = 4\ln(n+1),$$

o que conclui este prova . □

Finalmente, apresentamos dois corolários do teorema anterior.

Corolário 3.5 *O custo médio de busca de uma solução ótima para PBCM não é maior que* $4\ln(n+1)$.

Prova. Este resultado é uma conseqüência imediata do Teorema 3.4 e do fato de que uma ABB ótima não pode ter um custo maior que o da árvore construída pelo Algoritmo da Razão. □

Corolário 3.6 *O Algoritmo da Razão é* $4\ln(n+1)$*-aproximado.*

Prova. Este resultado é uma conseqüência imediata do Teorema 3.4 e do fato de que $\sum_{i=1}^{n} c(a_j) = 1$ é um limite inferior para o custo de uma ABB ótima. □

Vale mencionar que, para algumas estruturas de custo, o custo médio de busca de uma ABB ótima é $\Omega(\log n)$. Por exemplo, no caso onde todas as chaves têm custo $1/n$, a proposição 3.1 prova que este custo médio de busca não é menor do que $\log n - 1$. Observe que este custo difere do limite superior dado pelo Teorema 3.4 por um fator de

$$4\ln 2 + o(1) < 2.773 + o(1).$$

Por isto, podemos concluir que este limite superior é assintoticamente apertado a menos de um fator constante, como função apenas de n.

No entanto, existem exemplos onde o custo de uma ABB ótima é $O(1)$. Por exemplo, isto ocorre quando $c(a_i) = 2^{i-1}/(2^n - 1)$, para $i = 1, 2, \ldots, n$. Neste caso, observe que uma estratégia que sempre acessa primeiro a chave de menor custo do subvetor corrente tem um custo médio de busca dado por

$$\frac{1}{2^n - 1}\left(\sum_{i=1}^{n}(n-i+1)2^{i-1}\right) = 2 - \frac{n}{2^n - 1}.$$

Um problema que ainda está aberto consiste em determinar se existe alguma constante k tal que $c_R(\mathcal{A}) \leq k \times c^*(\mathcal{A})$ para qualquer estrutura de custos do vetor de entrada $\mathcal{A}$.

3.3.2 Análise de Desempenho

Agora, analisamos o desempenho assintótico do Algoritmo da Razão.

Proposição 3.7 *O Algoritmo da Razão executa em tempo* $O(nh_R)$

Prova. Claramente, a maior parte de esforço computacional realizado pelo Algoritmo da Razão corresponde às buscas pelas raízes das sub-árvores. Observe que cada chave a do vetor de entrada $\mathcal{A}$ é testada exatamente uma vez a cada ancestral de a escolhido como raiz. Como $\mathcal{A}$ tem exatamente n chaves, cada uma com até $h_R + 1$ ancestrais na árvore T_R, o Algoritmo da Razão executa em tempo $O(nh_R)$. □

Como $h_R \leq n$, o tempo de execução dado pela proposição anterior é $O(n^2)$ no pior caso.

3.3.3 Experimentos

Nesta subseção, relatamos alguns experimentos que comparam a qualidade das soluções obtidas por vários algoritmos encontrados na literatura para o PBCM, para duas estruturas de custo específicas. Estes experimentos sugerem que o Algoritmo da Razão é o melhor algoritmo não exato para estas estruturas de custo.

Consideramos os seguintes quatro algoritmos:

Exato: um algoritmo exato que roda em tempo $O(n^3)$ [11];

Razão: o Algoritmo da Razão (descrito nesta seção), que roda em tempo $O(nh_R)$;

ECBM: o Algoritmo ECBM (descrito na seção 3.4) com $\alpha = 2$, $K = 10$ e $t = \infty$, que roda em tempo linear;

Mínimo: uma estratégia gulosa proposta em [29], que sempre acessa primeiro a chave de menor custo do subvetor corrente;

BBin: uma busca binária.

Vale mencionar que o Algoritmo Mínimo foi proposto para uma aplicação específica em recuperação de textos [29]. Conseqüentemente, este algoritmo só obtém soluções de custo comparável ao ótimo para estruturas de custo com a seguinte propriedade: dado um subvetor qualquer de $\mathcal{A}$ com k

chaves, cada chave tem um custo mínimo neste subvetor com probabilidade $1/k$.

Utilizamos duas estruturas de custo em nossos experimentos. A primeira estrutura corresponde a uma aplicação em projetos de filtros [11]. Neste caso, o valor do custo $c(a_i)$ é proporcional a i^t, para $i = 1, \ldots, n$, onde t é uma constante positiva. Chamamos esta estrutura de custos de *custos polinomiais*. A segunda estrutura de custos é aleatória. Para $i = 1, \ldots, n$, o valor do custo $c(a_i)$ é proporcional a um valor sorteado no conjunto $\{1, \ldots, c^*\}$, onde cada valor pode ser sorteado com probabilidade $1/c^*$ independentemente dos outros sorteios. Neste caso, c^* é uma constante positiva. Chamamos esta estrutura de custos de *custos aleatórios*. Para manter a compatibilidade com os padrões adotados nesta tese, normalizamos todos vetores de custos de modo a garantir que $\sum_{i=1}^{n} c(a_i) = 1$.

Custos Polinomiais

A Tabela 3.2 descreve os custos das ABBs construídas para custos polinomiais, com $t = 1, 2, 3, 4$ e $n = 50, 200, 800$. Nesta tabela, os custos obtidos pelo algoritmo Exato são valores absolutos enquanto os custos obtidos pelos outros algoritmo são apresentados como diferenças percentuais em relação ao custo ótimo.

Tabela 3.2: Custos das ABBs construídas para custos polinomiais

t	1			2		
n	50	200	800	50	200	800
Exato	4.75	6.66	8.62	4.39	6.27	8.23
Razão (%)	2.15	1.63	1.27	1.83	1.27	0.97
ECBM (%)	6.55	6.02	5.01	4.80	3.97	3.16
BBin (%)	1.75	1.53	1.25	3.22	2.65	2.13

t	3			4		
n	50	200	800	50	200	800
Exato	4.09	5.94	7.89	3.85	5.67	7.61
Razão (%)	2.74	0.87	1.42	1.49	1.85	1.42
ECBM (%)	5.40	4.02	3.04	4.99	3.34	2.46
BBin (%)	3.59	2.92	2.35	3.50	2.88	2.33

Observe que todos os algoritmos obtêm soluções de boa qualidade, apresentando um erro de no máximo 7% em relação ao custo ótimo. Vale mencionar que o Algoritmo da Razão apresenta o melhor desempenho dentre

os algoritmos não exatos em todos os casos exceto para $t = 1$. Em todos os casos, ele obtém soluções com no máximo 3% de erro. Para $t = 1$, a busca binária apresenta um desempenho ligeiramente melhor do que o Algoritmo da Razão. Nesta tabela, omitimos os resultados relativos ao Algoritmo Mínimo porque esta estrutura de custos não tem a propriedade necessária para um desempenho razoável deste algoritmo. De fato, as soluções obtidas por este algoritmo para esta estrutura apresentam custos muitas vezes maiores do que os respectivos custos ótimos.

Custos Aleatórios

A Tabela 3.3 descreve os custos das ABBs construídas para custos aleatórios, com $c^* = 2, 5, 100, 1000$ e $n = 50, 200, 800$. Da mesma forma que na Tabela 3.2, os custos obtidos pelo algoritmo Exato são valores absolutos enquanto os custos obtidos pelos outros algoritmo são diferenças percentuais em relação ao custo ótimo. Para obter resultados mais estáveis, repetimos cada experimento uma centena de vezes, para cada par (c^*, n). A cada repetição, uma nova estrutura de custos foi sorteada. Neste caso, cada custo relatado representada uma média de cem custos obtidos para instâncias aleatórias diferentes.

Tabela 3.3: Custos das ABBs construídas para custos aleatórios

c^*	2			5		
n	50	200	800	50	200	800
Exato	3.81	5.07	6.39	3.01	3.67	4.34
Razão (%)	0.60	0.55	0.48	0.75	0.77	0.72
ECBM (%)	28.21	33.29	37.13	10.63	16.15	21.79
Mínimo (%)	1.01	0.84	0.76	3.21	2.92	2.43
BBin (%)	28.21	33.29	37.13	62.88	86.03	100.33

c^*	100			1000		
n	50	200	800	50	200	800
Exato	2.48	2.67	2.75	2.47	2.62	2.70
Razão (%)	0.47	0.48	0.52	0.48	0.49	0.52
ECBM (%)	2.41	2.93	3.75	1.94	2.56	2.48
Mínimo (%)	9.16	10.00	9.58	9.56	10.45	10.84
BBin (%)	97.93	154.07	215.27	96.97	161.70	220.99

Com relação aos resultados da Tabela 3.3, fazemos os seguintes comentários:

1. O custo ótimo esperado diminui à medida que c^* aumenta, pois c^* determina a razão máxima entre o maior e o menor custo de uma chave;

2. Em todos os casos, o Algoritmo da Razão obteve os menores custos médios dentre os algoritmos não exatos, sempre apresentando um erro médio abaixo de 1% em relação ao custo ótimo.

3. O algoritmo ECBM apresenta um comportamento idêntico ao da busca binária para $c^* = 2$, pois ele pode não fazer distinção entre dois custos que diferem por um fator menor ou igual a $\alpha = 2$. Neste caso, o ECBM apresenta soluções cujo erro relativo está entre 25% e 40%. Este erro relativo diminui sensivelmente à medida que o valor de c^* aumenta, chegando a menos de 3% para $c^* = 1000$.

4. O Algoritmo Mínimo apresenta soluções cujos custos médios são próximos aos obtidos pelo Algoritmo da Razão para $c^* = 2$, mas aumentam à medida que o valor de c^* aumenta, chegando a ter um erro relativo de 10% para $c^* = 1000$.

5. A busca binária tem um desempenho ruim para esta estrutura de custos. De fato, não é difícil provar que o custo esperado da busca binária é $\Theta(\log n)$ neste caso [29]. Conseqüentemente, o erro relativo desta estratégia aumenta à medida que c^* aumenta, o que está de acordo com os comentários do Item 1.

3.4 Algoritmos por Escala de Custos

Nesta seção, apresentamos os algoritmos de Escala de Custos para Busca Média (ECBM), para o PBCM, e de Escala de Custos para a Pior Busca (ECPB), para o PBPC.

De forma semelhante ao Algoritmo da Razão, tanto ECBM quanto o ECPB tentam balancear dois objetivos:

(i) produzir uma árvore razoavelmente balanceada;

(ii) acessar primeiro as chave de custo mais baixo.

Para isto, as chaves do vetor de entrada são agrupadas segundo uma escala de custos de acesso. Ao longo deste capítulo, dizemos que uma chave tem posto i quando ela pertencem ao i-ésimo grupo desta escala em ordem

crescente de custo. Então, ambos os algoritmos utilizam a *regra do menor posto* para construir uma ABB T, ou seja, as chaves de menor posto são sempre acessadas antes das de maior posto. Observe que esta regra persegue o objetivo (i). O objetivo (ii), por sua vez, é perseguido quando o algoritmo qual chave deve ser acessada primeiro dentre todas as chaves de posto mínimo do subvetor corrente.

Assim como o Algoritmo da Razão, ambos os algoritmos constroem uma ABB através de uma abordagem *top-down*. Primeiro, todas as chaves de posto mínimo do subvetor corrente são selecionadas. Além disso, cada subseqüência contígua de chaves que não foram selecionadas neste subvetor é condensada em um único nó. Neste caso, se as chaves a_i e a_{i+1} foram selecionadas, então assumimos que existe uma subseqüência vazia de chaves não selecionadas entre a_i e a_{i+1}. De forma semelhante, se a primeira (última) chave foi selecionada, então assumimos que existe uma subseqüência vazia de chaves não selecionadas antes (depois) desta chave. Cada subseqüência vazia mencionada anteriormente também é condensada em um nó. Em seguida, ambos os algoritmos constroem uma AEBB auxiliar (ver definição na seção 3.1) T_D, onde cada nó interno corresponde a uma chave selecionada e cada folha corresponde a uma subseqüência condensada. A única diferença entre o ECBM e o ECPB está no algoritmo utilizado para construir T_D.

Agora, observe que cada subseqüência condensada também corresponde a um subvetor do vetor de entrada. Por isto, ambos os algoritmos resolvem recursivamente os subproblemas associados a estas subseqüências. A ABB associada ao subvetor corrente é obtida após a substituição de cada folha de T_D pela ABB construída recursivamente para a subseqüência condensada correspondente. Mais adiante, descrevemos a construção de T_D para cada um dos dois algoritmos.

Agora, ilustramos o ECBM através de um exemplo. Seja $L = [a_1, \ldots, a_9]$ um subvetor do vetor de entrada cujos vetor de custos de acesso é

$$[0.1, 0.1, \mathbf{0.05}, \mathbf{0.03}, 0.3, 0.1, 0.08, 0.2, \mathbf{0.04}]. \tag{3-8}$$

Neste vetor, os custos das chaves de posto mínimo estão em negrito.

A Figura 3.2 representa alguns passos da construção de uma ABB T_L pelo ECBM, quando L é o subvetor corrente. Primeiro, o ECBM divide L em sete subseqüências de chaves: $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$ e $L_7 = []$. Observe que cada subseqüência de índice par (L_2, L_4 e L_6) contém exatamente uma chave. Cada chave contida

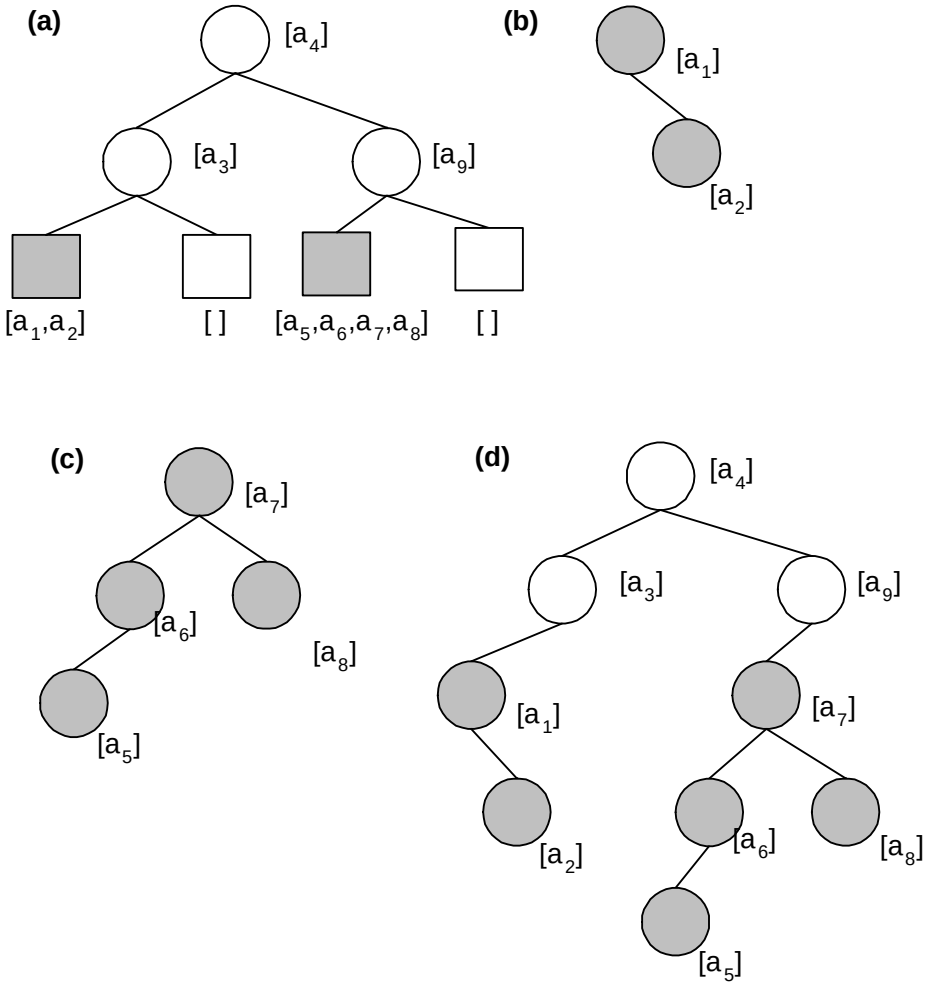

Figura 3.2: (a) Uma AEBB auxiliar T_D cujos nós internos estão associados às chaves a_3, a_4 e a_9. (b) Uma ABB construída pelo ECBM, para o subvetor $L_1 = [a_1, a_2]$. (c) Uma ABB construída pelo ECBM, para o subvetor $L_5 = [a_5, a_6, a_7, a_8]$. (d) Uma ABB T_L construída pelo ECBM, para o subvetor corrente L.

em uma destas subseqüências é associada a um nó interno de uma AEBB auxiliar T_D, representada pela Figura 3.2.(a). Além disso, as subseqüências de índice ímpar (L_1, L_3, L_5 e L_7) são condensadas e associadas às folhas de T_D. O ECBM constrói uma ABB para cada uma destas subseqüências através de chamadas recursivas. As Figuras 3.2.(b) e 3.2.(c) representam as ABB construídas para as subseqüências L_1 e L_5, respectivamente. Por último, o ECBM substitui cada folha de T_D pela ABB construída para a subseqüência correspondente. No caso das subseqüências vazias L_3 e L_7, as folhas correspondentes são removidas de T_D. A ABB resultante T_L é representada na Figura 3.2.(d).

Agora, formalizamos a noção de posto. Seja t um número inteiro positivo e α um fator de escala real e maior do que 1. Nas análises de

aproximação do ECBM e do ECPB apresentadas na seção 3.5, atribuímos valores apropriados a α e t. A escala de custos utilizada por estes algoritmos particiona o intervalo real $[0,1)$ em $t+1$ sub-intervalos de forma que $[0,1/\alpha^t)$ é o primeiro sub-intervalo e $[1/\alpha^{t-i+2}, 1/\alpha^{t-i+1})$ é o i-ésimo, para $i = 2,\ldots,t+1$. Com isto, definimos o posto de uma dada chave a_j como sendo igual a i quando o seu custo $c(a_j)$ está contido no i-ésimo sub-intervalo, para $i = 1,2,\ldots,t+1$. Além disso, definimos o posto de um subvetor L como sendo igual ao posto mínimo de uma chave contida em L. Denotamos o posto de L por $r(L)$. Por exemplo, observe que poderíamos ter $t = 4$ e $\alpha = 2$ para o vetor de custos dado por (3-8). Neste caso, como o primeiro sub-intervalo é $[0,1/16)$, apenas as chaves a_3, a_4 e a_9 têm posto igual a 1.

A Tabela 3.4 contém um pseudocódigo comum para o ECBM e para o ECPB. A única diferença entre os dois algoritmos é a construção da AEBB auxiliar T_D, que é realizada pela Função Caso-Médio no caso do ECBM e pela Função Pior-Caso no caso do ECPB. Observe que cada argumento da Função Caso-Médio fornece o número de chaves associadas a um nó (folha ou nó interno) da árvore que será construída. Por outro lado, os argumentos passados para a Função Pior-Caso são os custos (piores custos de busca) das ABB que substituirão as folhas de T_D. Vale mencionar que a Função Construir retorna, além da ABB construída para o subvetor L, o pior custo de busca desta ABB. Este último valor retornado não é utilizado quando o problema é o PBCM.

Para reduzir o fator de aproximação demonstrado para ambos os algoritmos, a Função Construir constrói uma ABB ótima para qualquer subvetor L com até K chaves ($|L| \leq K$), onde K é um parâmetro fixo. Esta árvore ótima sempre é construída através de um algoritmo por programação dinâmica em tempo $O(|L|^3)$. No caso do PBCM, utilizamos o algoritmo proposto em [11]. Este algoritmo utiliza a seguinte expressão:

$$c^*([a_1,\ldots,a_n]) = \min_{i=1,\ldots,n}\{nc(a_i) + c^*([a_1,\ldots,a_{i-1}]) + c^*([a_{i+1},\ldots,a_n])\}.$$

Já no caso do PBPC, modificamos este algoritmo para considerar o pior custo de busca ao invés do custo médio. Após esta modificação, obtemos seguinte expressão:

$$w^*([a_1,\ldots,a_n]) = \min_{i=1,\ldots,n}\{c(a_i) + \max\{w^*([a_1,\ldots,a_{i-1}]), w^*([a_{i+1},\ldots,a_n])\}.$$

Dadas estas expressões, os respectivos algoritmos exatos resultam de

Função Construir(L : vetor de chaves) : ABB e custo;
  Se $|L| \leq K$ **então**
    $T_L \leftarrow$ uma ABB ótima para L;
    retornar T_L e o seu pior custo de busca $w(T_L)$;
  Fim Se
  $a_{i_1}, a_{i_2}, \ldots, a_{i_k} \leftarrow$ as chaves de L cujo posto é $r(L)$;
  $i_0 \leftarrow 0$; $i_{k+1} \leftarrow |L| + 1$;
  Para $j = 1, \ldots, k$ **faça**
    $L_{2j} \leftarrow [a_{i_j}]$;
  Fim Para
  Para $j = 0, \ldots, k$ **faça**
    $L_{2j+1} \leftarrow [a_{i_j+1}, a_{i_j+2}, \ldots, a_{i_{j+1}-1}]$;
  Fim Para
  Para $j = 0, \ldots, k$ **faça**
    $(T_{L_{2j+1}}, w(T_{L_{2j+1}})) \leftarrow$ Construir(L_{2j+1});
  Fim Para
  Se o problema é o PBCM **então**
    $T_D \leftarrow$ Caso-Médio($|L_1|, |L_2|, \ldots, |L_{2k+1}|$);
  Senão
    $T_D \leftarrow$ Pior-Caso($w(T_{L_1}), w(T_{L_3}), \ldots, w(T_{L_{2k+1}})$);
  Fim Se
  Para $j = 1, \ldots, k$ **faça**
    associar a_{i_j} ao j-ésimo nó interno de T_D,
      na ordem dada por um percurso *em ordem*;
  Fim Para
  Para $j = 0, \ldots, k$ **faça**
    substituir a $(j+1)$-ésima folha de T_D por $T_{L_{2j+1}}$,
      na ordem dada por um percurso *em ordem*;
  Fim Para
  $T_L \leftarrow$ a ABB resultante;
  retornar T_L e $w(T_L)$;
Fim Função.

Tabela 3.4: Um pseudocódigo comum para o ECBM e para o ECPB

uma aplicação imediata da técnica de programação dinâmica [15].

3.4.1 A Função Caso-Médio

Para descrever esta função, definimos o problema conhecido como *Problema do Comprimento Ponderado de Caminho* (*Weighted Path Length Problem*). Dada uma lista de pesos $[w_1, \ldots, w_{2n+1}]$, o comprimento ponderado

de caminho de uma árvore estritamente binária T com $2n+1$ nós é definido como

$$\sum_{i=0}^{n} w_{2i+1} l_{2i+1} + \sum_{i=1}^{n} w_{2i}(l_{2i}+1),$$

onde l_j é o nível do j-ésimo nó visitado durante um percurso *em ordem* em T. Neste caso, observe que as folhas estão sempre associadas aos valores ímpares de j, enquanto os nós internos estão associados aos pares. O Problema do Comprimento Ponderado de Caminho consiste em encontrar, para uma dada lista de pesos, uma árvore estritamente binária cujo comprimento ponderado de caminho é mínimo. Existem vários algoritmos para este problema [2, 9, 17]. Vários destes algoritmos são baseados em critérios simples para escolha da raiz do subvetor corrente. Entre estes critérios, um dos mais simples é o Min-Max, que consistem em escolher como raiz o elemento de peso w_{2i} que minimiza

$$\max\left\{\sum_{j=1}^{2i-1} w_j, \sum_{j=2i+1}^{2n+1} w_j\right\}.$$

Vale mencionar que a construção de uma AEBB segundo este critério admite uma implementação que roda em tempo linear [5]. Pela sua simplicidade, a Função Caso-Médio utiliza esta implementação. Este algoritmo constrói uma árvore estritamente binária T_D para uma lista de pesos que contém a cardinalidade de cada subvetor L_i, ou seja, ele utiliza a lista de pesos $[|L_1|, |L_2|, \ldots, |L_{2k+1}|]$. Por exemplo, no caso do vetor de custos dado por (3-8), a Função Caso-Médio utiliza a lista de pesos $[2, 1, 0, 1, 4, 1, 0]$ para construir a árvore da Figura 3.2.(a).

3.4.2 A Função Pior-Caso

Em [26], Sebö e Waksman propõem um algoritmo para resolver o seguinte problema: dada uma lista de números $W = [w_1, w_2, \ldots, w_q]$, encontrar uma árvore binária T com q folhas que minimiza

$$f(T, W) = \max_{i=1,\ldots,q} \{l_i + w_i\},$$

onde l_i é o nível da i-ésima folha de T, na ordem dada por um percurso *em ordem*. Chamamos este problema de *Problema de Busca com Folgas*. Este algoritmo roda em tempo $O(q)$.

Seja T_D a árvore obtida pelo algoritmo proposto em [26], para a lista de números

$$W' = [w(L_1)\alpha^{t-r(L)+1}, w(L_3)\alpha^{t-r(L)+1}, \ldots, w(L_{2k+1})\alpha^{t-r(L)+1}].$$

Se $r(\mathcal{A}) > 1$, então a Função Pior-Caso recebe como argumentos os valores de $w(L_1), w(L_3), \ldots, w(L_{2k+1})$ e retorna a árvore T_D construída por este algoritmo. Vale mencionar que T_D também é uma árvore ótima para o seguinte problema: encontrar uma árvore binária T com $k+1$ folhas que minimiza

$$w'(T) = \max_{i=0,\ldots,k}\{l_{2i+1}/\alpha^{t-r(L)+1} + w(L_{2i+1})\}, \tag{3-9}$$

onde l_{2i+1} é o nível da i-ésima folha de T, na ordem dada por um percurso *em ordem*. Como $w'(T)$ é igual a $f(T, W')$ para qualquer T a menos de um fator multiplicativo $1/\alpha^{t-r(L)+1}$, qualquer solução T que minimiza $f(T, W')$ também minimiza $w'(T)$.

No caso em que $r(L) = 1$, a Função Pior-Caso retorna uma AEBB balanceada, ou seja, todas as folhas de T_D estão em nível menor ou igual a $\lceil \log(k+1) \rceil$.

3.5 Análise de Aproximação por Escala de Custos

Nesta seção, introduzimos uma nova técnica para analisar o fator de aproximação de algoritmos para problemas de busca com custos de acesso variados. Com base nesta técnica, analisamos os algoritmos ECBM e ECPB, propostos na seção 3.4 para o PBCM e para o PBPC, respectivamente. Provamos que ambos os algoritmos constroem soluções $(2 + \epsilon + o(1))$-aproximadas, para qualquer $\epsilon > 0$ fixo.

3.5.1 Técnica de Partição

Agora, introduzimos uma técnica para obtenção de limites inferiores para problemas de busca que se aplica tanto ao PBCM quanto ao PBPC. Chamamos esta técnica de *Técnica de Partição*.

Seja $L = [a_1, \ldots, a_m]$ um vetor de chaves e T uma ABB para L. Além disso, considere uma partição genérica do vetor L em k subseqüências de

chaves contíguas e não vazias denotadas por $[a_1, \ldots, a_{i_1}]$, $[a_{i_1+1}, \ldots, a_{i_2}], \ldots,$ $[a_{i_{k-1}+1}, \ldots, a_m]$. Denotamos por L_j o subvetor

$$[a_{i_{j-1}+1}, \ldots, a_{i_j}],$$

associada à j-ésima subseqüência da partição de L, onde $i_0 = 0$ e $i_k = m$. A Técnica de Partição é uma abordagem que pode ser utilizada para obter limites inferiores para o custo médio (pior custo) de busca de uma ABB ótima para L como função dos custos médios (piores custos) de busca de ABBs ótimas para os subvetores $L_1, \ldots, L_k$. Este técnica utiliza o seguinte lema.

Lema 2 *Para cada $i \in \{1, \ldots, k\}$, existe uma chave no subvetor L_i que é ancestral em T de todas as chaves de L_i.*

Prova. Se $|L_i| = 1$, então a corretude deste lema é trivial. Logo, assumimos que $|L_i| > 1$. Neste caso, seja a uma chave de L_i com o número máximo de descendentes em T. Provamos por absurdo que a é ancestral em T de todas as chaves de L_i. Primeiro, suponha que existe uma chave b em L_i que não é descendente de a. Então, temos dois casos: b é ancestral de a ou b não é ancestral de a. Se b é ancestral de a, então b tem mais descendentes em T do que a, o que contradiz a nossa suposição inicial. Absurdo. No segundo caso, se b não é ancestral de a, então seja c o ancestral comum de a e b que tem o nível máximo em T. Neste caso, observe que a e b não podem ser descendentes do mesmo filho de c. Em caso contrário, o critério utilizado na escolha de c daria preferência a este filho de c. Como a e b pertencem a sub-árvores diferentes de c, concluímos que a chave c está posicionada entre as chaves a e b no vetor L. Logo, como $a, b \in L_i$ e L_i corresponde a uma subseqüência contígua de chaves, concluímos que c também pertencem a L_i. Portanto, o fato de que c tem mais descendentes em T do que a também contradiz a nossa suposição inicial. Absurdo. Daí concluímos que a é ancestral em T de todas as chaves de L_i. □

Se uma chave a de L_i é ancestral em T de todas as chaves de L_i, então dizemos que a é o *pivô* de L_i em T. Dada uma chave $a \in L_i$, se a não é o pivô de L_i em T, então definimos $\phi(a)$ como o ancestral estrito de a que pertence a L_i e tem nível máximo em T. Em caso contrário, se a é o pivô de L_i em T, então $\phi(a) = a$. Com isto, o procedimento apresentado a seguir constrói uma árvore T_i para as chaves do subvetor L_i a partir de T.

Procedimento Partição(T, L_i)

Para toda chave $a \in L_i$ tal que $\phi(a) \neq a$, **faça**
Se a é descendente do filho à esquerda de $\phi(a)$ em T **então**
conectar a como o filho à esquerda de $\phi(a)$ em T_i;
Senão
conectar a como o filho à direita de $\phi(a)$ em T_i;
Fim Se
Fim Para

Por exemplo, seja $L = [a_1, \ldots, a_{11}]$, $L_1 = [a_1, a_2, a_3]$, $L_2 = [a_4, a_5, a_6, a_7, a_8]$ e $L_3 = [a_9, a_{10}, a_{11}]$. Neste caso, a Figura 3.3.(a) mostra uma ABB T para L. As árvores T_1, T_2 e T_3 construídas pelo Procedimento Partição a partir de T são representadas pelo Figura 3.3.(b). Nestas figuras, o número que aparece dentro de cada nó representa o índice da chave correspondente. Além disso, tanto os nós de T que pertencem a L_2 quanto os nós de T_2 estão pintados de cinza na Figura 3.3. Para este exemplo, observe que os pivôs de L_1, L_2 e L_3 em T são a_2, a_8 e a_{10}, respectivamente.

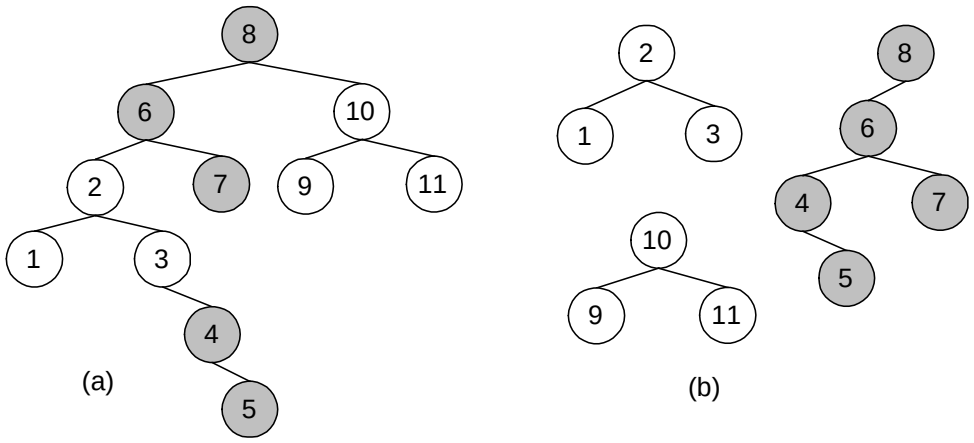

Figura 3.3: (a) Uma ABB T para o vetor $[a_1, \ldots, a_{11}]$. (b) As árvores T_1, T_2 e T_3 construídas para os subvetores $L_1 = [a_1, a_2, a_3]$, $L_2 = [a_4, \ldots, a_8]$ e $L_3 = [a_9, a_{10}, a_{11}]$, respectivamente.

A seguir, provamos que as árvores construídas pelo Procedimento Partição são sempre ABBs. Para isto, temos o seguinte lema.

Lema 3 *Dados um vetor L, uma ABB T para este vetor e um subvetor L_i associado a uma subseqüência contígua de chaves de L, se T_i é a árvore construída pela execução de Partição(T, L_i), então T_i é uma ABB.*

Prova. Demonstramos este lema provando por indução sobre o número de chaves de L_i que T_i é uma ABB cuja raiz é o pivô de L_i em T.

Se $|L_i| = 1$, a afirmação anterior vale pois um nó isolado é uma ABB. Agora, assumimos que esta afirmação vale para qualquer subvetor de L com menos do que k chaves. Então, provamos esta afirmação para o caso onde $|L_i| = k$.

Seja a o pivô de L_i. Sejam também L_i^1 e L_i^2 os subvetores de L_i à esquerda e à direita da chave a, respectivamente. Além disso, sejam T_i^1 e T_i^2 as árvores construídas pelas execuções de Partição(T, L_i^1) e Partição(T, L_i^2), respectivamente. Pela hipótese indutiva, T_i^1 e T_i^2 são ABBs cujas raízes são os pivôs de L_i^1 e L_i^2, respectivamente. Como a é ancestral de todos os nós de T que pertencem a L_i e as chaves de L_i^1 (L_i^2) estão todas à esquerda (direita) de a, concluímos que nenhum nó de T que pertence a L_i^1 é ancestral ou descendente de um nó que pertence a L_i^2. Daí, concluímos que nenhum nó de T_i que pertence a L_i^1 é pai ou filho de um nó que pertence a L_i^2. Também observamos que apenas o pivô de L_i^1 (L_i^2) pode ser filho à esquerda (direita) de a em T_i, pois, para qualquer outro nó de T que pertence a L_i^1 (L_i^2), o pivô de L_i^1 (L_i^2) é um ancestral que tem maior nível em T. Logo, para uma dada chave b de L_i^1 (L_i^2) exceto o pivô, quando o Procedimento Partição escolhe o pai de b em T_i ele escolhe obrigatoriamente um nó que também pertence a L_i^1 (L_i^2). Portanto, o nó escolhido é o mesmo que na construção de T_i^1 (T_i^2). Conseqüentemente, as ABBs T_i^1 e T_i^2 são sub-árvores de T_i. Como as raízes de T_i^1 e T_i^2 (os pivôs de L_i^1 e L_i^2) são os filhos à esquerda e à direita de a, respectivamente, concluímos que T_i é uma ABB cuja raiz é a. □

Observe que, para qualquer estrutura de custos, a soma dos custos médios de busca das árvores T_1, T_2 e T_3 representadas pela Figura 3.3.(b) não é maior que o custo médio de busca de T. Isto sempre ocorre com as árvores construídas pelo Procedimento Partição porque, por construção, qualquer ancestral de uma dada chave a em T_i também é ancestral de a em T. De forma semelhante, observamos que o pior custo de busca de T não é menor do que o maior dentre os piores custos de busca de T_1, T_2 e T_3. Com base nestas observações, a Técnica de Partição permite obter limites inferiores para o custo médio e para o pior custo de busca de uma ABB ótima da seguinte forma. Por exemplo, suponha que a ABB T apresentada na Figura 3.3.(a) é uma solução ótima para o PBCM(L). Como T_i é uma solução viável para o PBCM(L_i), para $i = 1, 2, 3$, concluímos que o custo médio de busca de T_i não pode ser menor do que $c^*(L_i)$. Daí, obtemos que

$$c^*(L) \geq c^*(L_1) + c^*(L_2) + c^*(L_3). \tag{3-10}$$

De forma semelhante, também obtemos que

$$w^*(L) \geq \max\{w^*(L_1), w^*(L_2), w^*(L_3)\}.$$

O lema apresentado a seguir generaliza estes limites inferiores.

Lema 4 *Seja L um vetor particionado em k subvetores $L_1, L_2, \ldots, L_k$, cada um deles associado a uma subseqüência não vazia e contígua de chaves de L. Neste caso, temos*

$$c^*(L) \geq \sum_{i=1}^{k} c^*(L_i)$$

e

$$w^*(L) \geq \max_{i=1,\ldots,k}\{w^*(L_i)\}.$$

Prova. Segue da discussão anterior. □

Apesar destes limites inferiores não serem suficientes para demonstrar a existência de fatores de aproximação constante para o ECBM e para o ECPB, a mesma abordagem pode ser estendida para obter limites melhores. Nas subseções seguintes, mostramos como melhorar estes limites inferiores utilizando características específicas de cada função de custo.

Para isto, assumiremos sempre que um dado subvetor L recebido como argumento pela Função Construir é particionado em $2k+1$ subvetores $L_1, L_2, \ldots, L_{2k+1}$, cada um deles associado a uma subseqüência contígua de chaves de L. Ambos os algoritmos realizam este particionamento nos dois primeiros laços do pseudocódigo apresentado na Figura 3.4. Vale enfatizar que esta partição de L não inclui as subpartições realizadas nas chamadas recursivas à Função Construir. Além disso, definimos

$$I(L) = \{i| \ |L_i| > 0 \text{ e } r(L_i) > r(L)\}$$

e

$$I'(L) = \{2, 4, \ldots, 2k\}.$$

3.5.2
Extensão da Técnica de Partição para o PBCM

Aqui, estendemos a Técnica de Partição apresentada anteriormente a fim de aprimorar o limite inferior para o custo médio de busca de uma ABB ótima dado pelo Lema 4. O limite inferior obtido é função de um parâmetro real d, que será discutido mais adiante.

Para isto, definimos uma instância paramétrica do PBCM. Dados um número real d maior ou igual a $\min_{a\in L}\{c(a)\}$ e um vetor L, denotamos por PBCM(L,d) uma instância do PBCM tal que o custo da chave a é $c(a)-d$, para todo $a \in L$. Com isto, o custo médio de busca de uma ABB T para o PBCM(L,d) é dado por

$$c_d(T) = \sum_{a_i\in L}\sum_{x\in N(a_i,T)} (c(x)-d).$$

A idéia chave para aprimorar o limite inferior dado pelo Lema 4 é decompor o custo $c(a)$ de cada chave $a \in L$ em duas parcelas: uma igual a d e outra igual a $c(a)-d$. A partir desta decomposição, mostramos um limite inferior para $c^*(L)$ pode ser escrito como a soma de dois limites inferiores distintos: um para o PBCM(L,d) e outro para uma instância particular do PBCM(L,d) onde todas as chaves têm custo igual a d. O próximo teorema apresenta uma formalização desta idéia.

Antes, porém, introduzimos mais alguns definições. Denotamos por $T_L^*(d)$ uma ABB ótima para o PBCM(L,d). Para simplificar, denotamos o custo médio de busca desta árvore por $c^*(L,d)$ ao invés de $c_d(T_L^*(d))$. No caso em que $d=0$, omitimos o argumento d, obtendo $c^*(L)=c^*(L,0)$.

Teorema 3.8 *Sejam d_1 e d_2 dois números reais não negativos tais que*

$$d_1 + d_2 \leq \min_{a\in L}\{c(a)\}.$$

Neste caso, sempre temos

$$c^*(L,d_1) \geq d_2 \times |L|(\log|L|-1) + \sum_{i\in I(L)\cup I'(L)} c^*(L_i, d_1+d_2)$$

Prova. Seja a uma chave contida no subvetor L. Denotamos por $l(a)$ o nível do nó a na árvore $T_L^*(d_1)$ e por $N_L(a)$ o conjunto de todos os ancestrais de a nesta árvore. Como $T_L^*(d_1)$ é uma solução viável para o PBCM(L, d_1+d_2),

o custo médio de busca desta árvore para esta instância não pode ser menor do que $c^*(L, d_1 + d_2)$. Daí, obtemos

$$c^*(L, d_1) = \sum_{a \in L} \sum_{x \in N_L(a)} d_2 + \sum_{a \in L} \sum_{x \in N_L(a)} (c(x) - d_1 - d_2) \quad (3\text{-}11)$$

$$\geq \sum_{a \in L} d_2(l(a) + 1) + c^*(L, d_1 + d_2). \quad (3\text{-}12)$$

Além disso, pelo Lema 4, temos

$$c^*(L, d_1 + d_2) \geq \sum_{i \in I(L) \cup I'(L)} c^*(L_i, d_1 + d_2) \quad (3\text{-}13)$$

Como $T_L^*(d_1)$ também é uma solução viável para uma instância do PBCM(L) onde todos os custos são iguais a d_2, pela proposição 3.1, o custo médio de busca desta árvore para esta instância não pode ser menor do que $d_2|L|(\log|L| - 1)$. Logo, temos

$$d_2 \sum_{a \in L} (l(a) + 1) \geq d_2|L|(\log|L| - 1).$$

Portanto, combinando a desigualdade anterior com (3-12) e (3-13), concluímos que

$$c^*(L, d_1) \geq d_2|L|(\log|L| - 1) + \sum_{i \in I(L) \cup I'(L)} c^*(L_i, d_1 + d_2).$$

□

Por exemplo, aplicando o teorema anterior à partição representada pela Figura 3.3, obtemos

$$c^*(L, d_1) \geq 27d_2 + c^*(L_1, d_1 + d_2) + c^*(L_2, d_1 + d_2) + c^*(L_3, d_1 + d_2).$$

Em particular, assumindo que $d_1 = d_2 = 0$, obtemos o mesmo limite inferior dado por (3-10).

Vale mencionar que o principal limite inferior utilizado na análise do Algoritmo ECBM aplica recursivamente o resultado do teorema anterior para todos os subvetores de $\mathcal{A}$ passados como argumento para a Função Construir. Este limite inferior é estabelecido mais adiante no Teorema 3.10.

No próximo teorema, apresentamos uma desigualdade que relaciona os valores de $c^*(L)$ e $c^*(L, d_1)$. Esta desigualdade é útil para demonstrar um fator de aproximação constante para o ECBM, no Teorema 3.14.

Teorema 3.9 *Sejam d_1 e d_2 tais que*

$$0 \leq d_1 < d_2 \leq \min_{a \in L}\{c(a)\}.$$

Neste caso, sempre temos

$$\frac{c^*(L)}{c^*(L, d_1)} \leq \frac{d_2}{d_2 - d_1}.$$

Prova. Seja a uma chave contida no subvetor L. Denotamos por $N_L(a)$ o conjunto de todos os ancestrais de a na árvore $T_L^*(d_1)$. Por definição, temos

$$c^*(L, d_1) = \sum_{a \in L} \sum_{x \in N_L(a)} (c(x) - d_1).$$

Como $T_L^*(d_1)$ é uma solução viável para o PBCM$(L, 0)$, o custo médio de busca desta árvore para esta instância não pode ser menor do que $c^*(L)$, ou seja, sempre temos

$$c^*(L) \leq \sum_{a \in L} \sum_{x \in N_L(a)} c(x).$$

Agora, seja $a^* = \arg\min_{a \in L}\{c(a)\}$. Obtemos a partir da desigualdade anterior que

$$\begin{aligned} \frac{c^*(L)}{c^*(L, d_1)} &\leq \frac{\displaystyle\sum_{a \in L} \sum_{x \in N_L(a)} c(x)}{\displaystyle\sum_{a \in L} \sum_{x \in N_L(a)} (c(x) - d_1)} \\ &\leq \max_{a \in L} \left\{ \frac{c(a)}{c(a) - d_1} \right\} = \frac{c(a^*)}{c(a^*) - d_1}, \end{aligned}$$

Como $d_2 \leq c(a^*)$, concluímos que

$$\frac{c^*(L)}{c^*(L, d_1)} \leq \frac{c(a^*)}{c(a^*) - d_1} \leq \frac{d_2}{d_2 - d_1}$$

□

3.5.3
Extensão da Técnica de Partição para o PBPC

Aqui, estendemos a Técnica de Partição apresentada anteriormente a fim de aprimorar o limite inferior para o pior custo de busca de uma ABB ótima dado pelo Lema 4. Para isto, analisamos os níveis dos pivôs dos subvetores numa ABB ótima $T^*_W(L)$ para o PBPC(L). Por exemplo, no caso da Figura 3.3, o pior custo de busca da ABB T para o subvetor L não pode ser menor do que o pior custo de busca da ABB T_1 para o subvetor L_1. Este limite inferior pode ser aprimorado da seguinte forma. Seja d um número real não negativo menor ou igual a $\min_{a\in L}\{c(a)\}$. Como o nó de número 2, que é o pivô de L_1, tem nível 2 em T, podemos concluir que o pior custo de busca de T também não pode ser menor do que o pior custo de busca de T_1 somado com $2d$. O lema que apresentamos a seguir formaliza esta idéia.

Antes, porém, introduzimos mais alguns definições. Para $j \in I(L) \cup I'(L)$, seja y_j o pivô do subvetor L_j na árvore $T^*_W(L)$. Seja também l^W_j o nível do nó y_j na árvore $T^*_W(L)$.

Lema 5 *Seja d um número real tal que*

$$0 \leq d \leq \min_{a\in L}\{c(a)\}.$$

Neste caso, sempre temos

$$w^*(L) \geq \max_{i\in I(L)\cup I'(L)}\{d \times l^W_i + w^*(L_i)\}. \tag{3-14}$$

Prova. Seja a uma chave do subvetor L_i. Denotamos por $N_L(a)$ o conjunto de todos os ancestrais do nó a na árvore $T^*_W(L)$. O nó a tem dois tipos de ancestral em $T^*_W(L)$: aqueles cujas chaves estão em L_i e aqueles cujas chaves não estão em L_i. Como as chaves associadas aos ancestrais estritos de y_i em $T^*_W(L)$ não estão em L_i, podemos concluir que o custo de buscar a segundo a árvore $T^*_W(L)$ não é menor do que

$$d \times l^W_i + \sum_{x\in(N_L(a)\cap L_i)} c(x).$$

Logo, o pior custo de busca de uma chave que está em L_i segundo a árvore $T^*_W(L)$ não é menor do que

$$d \times l^W_i + \max_{a\in L_i}\left\{\sum_{x\in(N_L(a)\cap L_i)} c(x)\right\}. \tag{3-15}$$

Observe que a segunda parcela de (3-15) é igual ao pior custo de busca da árvore T_i construída pelo Procedimento Partição$(T^*_W(L), L_i)$. Como T_i é uma solução viável para o PBPC(L_i), concluímos que o pior custo de busca da árvore T_i não é menor do que $w^*(L_i)$. Portanto, o pior custo de busca de uma chave que está em L_i segundo a árvore $T^*_W(L)$ não é menor do que

$$d \times l_i^W + w^*(L_i).$$

Daí, estabelecemos o resultado enunciado por este lema tomando o máximo dentre os limites inferiores associados aos subvetores $\{L_i |\ i \in I(L) \cup I'(L)\}$. □

Mais adiante, ao analisar o Algoritmo ECPB atribuímos valores a d de acordo com o posto do subvetor atual L. Precisamente, fazemos $d = 1/\alpha^{t-r(L)+2}$ quando $r(L) > 1$ e $d = 0$ em caso contrário. Vale mencionar que estas escolhas são adequadas no sentido de satisfazer à hipótese $d \leq \min_{a \in L}\{c(a)\}$, utilizada pelo Lema 5.

3.5.4
Análise do Algoritmo ECBM

Nesta seção, analisamos o fator de aproximação do Algoritmo ECBM.

Para isto, introduzimos mais algumas idéias e definições. Seja Z o conjunto de todos o subvetores não vazios de $\mathcal{A}$ que são passados como argumento para a Função Construir em algum momento durante a execução de Construir$(\mathcal{A})$. Além disso, seja Z_K o conjunto de todos subvetores contidos em Z com mais de K chaves. Vale lembrar que, a Função Construir constrói uma ABB ótima para todo subvetor passado como argumento com até K chaves.

Em nossa análise, estabelecemos limites inferiores para uma ABB ótima e o limites superiores para a ABB construída pelo ECBM. A análise de aproximação do ECBM é realizada através de comparações entre estes dois tipos de limites. Estas comparações são possíveis porque tanto o principal limite inferior, estabelecido pelo Teorema 3.10, quanto o principal limite superior, estabelecido pelo Teorema 3.13, são escritos como função do posto e do número de chaves de cada subvetor de Z.

Dados um vetor de entrada $\mathcal{A}$ e o conjunto Z obtido através da execução de Construir$(\mathcal{A})$, definimos uma árvore T(Construir) da seguinte forma. Cada nó de T(Construir) está associado a um subvetor de Z. O nó associado a $\mathcal{A}$ é a raiz de T(Construir). Para um dado subvetor $L \in Z$, os nós

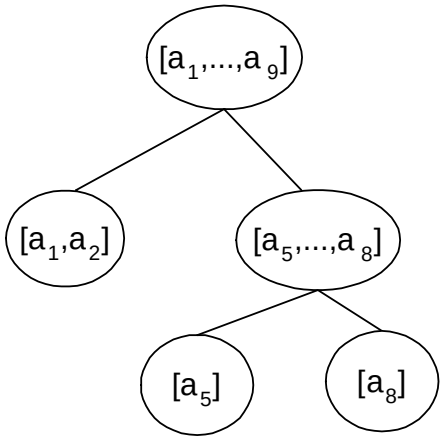

Figura 3.4: Uma árvore T(Construir)

filhos de L em T(Construir) estão associados aos subvetores $\{L_i | i \in I(L)\}$ de L. Observe que os subvetores $L_2, L_4, \ldots, L_{2k}$ não são passados como argumento para a Função Construir. Por isto, eles não pertencem a Z e não estão associados a nenhum nó de T(Construir).

Por exemplo, sejam um vetor de entrada $\mathcal{A} = [a_1, \ldots, a_9]$ e um vetor de custos associado $[0.1, 0.1, 0.05, 0.03, 0.3, 0.1, 0.08, 0.2, 0.04]$. Além disso, suponha que $\alpha = 2$, $t = 4$ e $K = 2$. Neste caso, a Figura 3.4 representa uma árvore T(Construir) obtida através da execução de Construir($\mathcal{A}$). Observe que, para esta árvore, temos $Z_2 = \{[a_1, \ldots, a_9], [a_5, \ldots, a_8]\}$ e $Z - Z_2 = \{[a_1, a_2], [a_5], [a_8]\}$.

Agora, apresentamos uma visão geral da utilização da árvore T(Construir) ao longo desta subseção. Observe que o Teorema 3.8 fornece um limite inferior para $c^*(L)$ como função dos custos de ABBs ótimas para os subvetores de L. De forma semelhante, o Corolário 3.12, que será apresentado mais adiante, fornecerá um limite superior para $c(L)$, como função dos custos de ABBs construídas pelo ECBM para estes mesmos subvetores. Por isto, uma comparação entre estes limites inferior e superior precisaria de alguma hipótese indutiva relacionando os custos destas ABBs. Vale lembrar que os subvetores de L estão associados aos filhos do nó L na árvore T(Construir). Portanto, se avaliarmos cada um destes limites recursivamente ao longo de T(Construir), obteremos novos limites que só dependem dos custos de ABBs para os subvetores de $\mathcal{A}$ associados às folhas de T(Construir). Esta abordagem será utilizada nas provas dos Teoremas 3.13 e 3.10. Vale lembrar que os subvetores associados às folhas de T(Construir) não têm mais do que K chaves. Como o ECBM sempre constrói ABBs ótimas para estes subvetores, a comparação entre os novos limites inferior e

superior permitirá provar um fator de aproximação constante.

Ao longo desta subseção, denotamos por $g(L)$ o posto do vetor associado ao pai do nó L na árvore T(Construir). Logo, se o nó L é filho do nó L' em T(Construir), então $g(L) = r(L')$. Para o vetor de entrada $\mathcal{A}$, definimos que $g(\mathcal{A}) = -\infty$. Finalmente, denotamos por T_L a ABB construída pelo ECBM para o subvetor L.

Limite Inferior

Aqui, demonstramos um limite inferior para o custo médio de busca de uma ABB ótima, para o vetor de entrada $\mathcal{A}$.

Para isto, aplicamos recursivamente o limite inferior dado pelo Teorema 3.8 ao longo da árvore T(Construir), começando pelo nó $\mathcal{A}$. Antes porém, definimos

$$R(L) = \begin{cases} 0, \text{ caso } r(L) = 1 \\ 1/\alpha^{t-r(L)+2}, \text{ caso } r(L) > 1 \end{cases}$$

Observe que $R(L)$ é um limite inferior para o custo de qualquer chave de L, ou seja,

$$R(L) \leq \min_{a \in L}\{c(a)\}.$$

Também definimos $G(L)$ da seguinte forma. Se o nó L' é o pai do nó L na árvore T(Construir), então $G(L) = R(L')$. Para o vetor de entrada $\mathcal{A}$, definimos $G(\mathcal{A}) = 0$.

O próximo teorema fornece o limite inferior mencionado anteriormente.

Teorema 3.10

$$\max\left\{1, \sum_{L \in Z_K} (R(L) - G(L))|L|(\log|L| - 1) + \sum_{L \in (Z - Z_K)} c^*(L, G(L))\right\}$$

é um limite inferior para $c^*(\mathcal{A})$.

Prova. Demonstramos este teorema por indução sobre os níveis da árvore T(Construir). Para isto, definimos Z^i e Z_K^i da seguinte forma:

$$Z^i = \{L|\ L \in Z \text{ e o nível do nó } L \text{ em } T(\text{Construir}) \text{ é} \leq i\ \}$$

e

$$Z_K^i = \{L|\ L \in Z^i \text{ e } |L| > K\}.$$

Neste caso, basta provar que

$$c^*(\mathcal{A}) \geq 1 \tag{3-16}$$

e que

$$c^*(\mathcal{A}) \geq \sum_{L \in Z_K^l} (R(L) - G(L))\,|L|(\log|L| - 1) + \sum_{L \in (Z^{l+1} - Z_K^l)} c^*(L, G(L)), \tag{3-17}$$

para qualquer número natural l. A expressão deste teorema é obtida tomando-se o máximo entre (3-16) e (3-17) e fazendo l igual à altura da árvore T(Construir). Neste caso, temos $Z_K^l = Z_K$ e $Z^l = Z^{l+1} = Z$.

Como $\sum_{i=1}^{n} c(a_i) = 1$, $c^*(\mathcal{A})$ não pode ser menor do que 1. Isto prova a validade de (3-16).

Agora, provamos a validade de (3-17) por indução sobre o valor de l. Para $l = 0$, se $|\mathcal{A}| \leq K$, então (3-17) decorre do fato de que $Z^1 = \{\mathcal{A}\}$, $Z_K^0 = \emptyset$ e $G(\mathcal{A}) = 0$. Substituindo-se estes valores em (3-17), obtemos $c^*(\mathcal{A}) \geq c^*(\mathcal{A})$, o que é óbvio. Logo, assumimos que $|\mathcal{A}| > K$. Primeiro, observamos que $Z_K^0 = \{\mathcal{A}\}$ e definimos $\{\mathcal{A}_i | i \in I(\mathcal{A})\} = Z^1 - Z_K^0$. Neste caso, precisamos provar que

$$c^*(\mathcal{A}) \geq R(\mathcal{A}) \times |\mathcal{A}|(\log|\mathcal{A}| - 1) + \sum_{i \in I(\mathcal{A})} c^*(\mathcal{A}_i, G(\mathcal{A}_i))$$

Observe que $G(\mathcal{A}_i) = R(\mathcal{A}))$. Além disso, como $G(\mathcal{A}) = 0$, temos que

$$R(\mathcal{A}) + G(\mathcal{A}) \leq \min_{a \in \mathcal{A}}\{c(a)\}.$$

Aplicando o Teorema 3.8 para o vetor de entrada $\mathcal{A}$, com $d_1 = G(\mathcal{A}) = 0$ e $d_2 = R(\mathcal{A})$, obtemos

$$\begin{aligned} c^*(\mathcal{A}) &\geq R(\mathcal{A})|\mathcal{A}|(\log|\mathcal{A}| - 1) + \sum_{i \in I'(\mathcal{A})} c^*(\mathcal{A}_i, R(\mathcal{A})) + \sum_{i \in I(\mathcal{A})} c^*(\mathcal{A}_i, R(\mathcal{A})) \\ &\geq R(\mathcal{A})|\mathcal{A}|(\log|\mathcal{A}| - 1) + \sum_{i \in I(\mathcal{A})} c^*(\mathcal{A}_i, G(\mathcal{A}_i)). \end{aligned}$$

Portanto, (3-17) vale para $l = 0$.

Agora, assumindo que o resultado vale para $l = p$ provamos que ele vale para $l = p + 1$. Aplicando a hipótese indutiva, obtemos

$$c^*(\mathcal{A}) \geq \sum_{L \in Z_K^p} (R(L) - G(L))|L|(\log|L| - 1) + \sum_{L \in (Z^{p+1} - Z_K^p)} c^*(L, G(L))$$

Basicamente, o passo indutivo consiste em expandir as parcelas do lado direito da desigualdade anterior que correspondem aos subvetores que estão em nível $p + 1$ em T(Construir) e têm mais de K chaves. Para isto, aplicamos o Teorema 3.8. Como o conjunto $Z_K^{p+1} - Z_K^p$ contém exatamente estes subvetores, reescrevemos o conjunto $Z^{p+1} - Z_K^p$ como a união de dois conjuntos disjuntos dados respectivamente por $Z^{p+1} - Z_K^{p+1}$ e $Z_K^{p+1} - Z_K^p$. Daí, concluímos que

$$\begin{aligned} c^*(\mathcal{A}) \quad \geq \quad & \sum_{L \in Z_K^p} (R(L) - G(L))|L|(\log|L| - 1) + \sum_{L \in (Z^{p+1} - Z_K^{p+1})} c^*(L, G(L)) \\ & + \sum_{L \in (Z_K^{p+1} - Z_K^p)} c^*(L, G(L)) \end{aligned} \tag{3-18}$$

Seja $L \in (Z_K^{p+1} - Z_K^p)$. Observe que $G(L_i) = R(L)$. Além disso, como $R(L) \leq \min_{a_i \in L}\{c(a_i)\}$, concluímos que $G(L) + (R(L) - G(L)) \leq \min_{a_i \in L}\{c(a_i)\}$. Daí, aplicando o Teorema 3.8 para o subvetor L, com $d_1 = G(L)$ e $d_2 = R(L) - G(L)$, obtemos

$$\begin{aligned} c^*(L, G(L)) \quad \geq \quad & (R(L) - G(L))|L|(\log|L| - 1) + \sum_{i \in I'(L)} c^*(L_i, R(L)) \\ & + \sum_{i \in I(L)} c^*(L_i, R(L)) \\ \geq \quad & (R(L) - G(L))|L|(\log|L| - 1) + \sum_{i \in I(L)} c^*(L_i, G(L_i)). \end{aligned}$$

Somando a desigualdade anterior para todo subvetor $L \in (Z_K^{p+1} - Z_K^p)$, obtemos

$$\sum_{L \in (Z_K^{p+1} - Z_K^p)} c^*(L, G(L)) \quad \geq \quad \sum_{L \in (Z_K^{p+1} - Z_K^p)} (R(L) - G(L))|L|(\log|L| - 1)$$

$$+ \sum_{L \in (Z^{p+2} - Z^{p+1})} c^*(L, G(L)). \qquad (3\text{-}19)$$

Substituindo-se a última parcela do lado direito de (3-18) pelo lado direito de (3-19) e utilizando o fato de que $Z^{p+2} - Z^{p+1}$ e $Z^{p+1} - Z_K^{p+1}$ são conjuntos disjuntos cuja união é o conjunto $Z^{p+2} - Z_K^{p+1}$, obtemos

$$\begin{aligned} c^*(\mathcal{A}) \;\geq\; & \sum_{L \in Z_K^p} (R(L) - G(L))|L|(\log|L| - 1) + \sum_{L \in (Z^{p+1} - Z_K^{p+1})} c^*(L, G(L)) \\ & + \sum_{L \in (Z_K^{p+1} - Z_K^p)} (R(L) - G(L))|L|(\log|L| - 1) \\ & + \sum_{L \in (Z^{p+2} - Z^{p+1})} c^*(L, G(L)) \\ = \; & \sum_{L \in Z_K^{p+1}} (R(L) - G(L))|L|(\log|L| - 1) + \sum_{L \in (Z^{p+2} - Z_K^{p+1})} c^*(L, G(L)), \end{aligned}$$

o que completa esta prova. □

Limite Superior

Aqui, demonstramos um limite superior para o custo médio de busca $c(\mathcal{A})$ de uma ABB $T_{\mathcal{A}}$ construída pelo ECBM, para o vetor de entrada $\mathcal{A}$.

Seja L um subvetor que pertence a Z. Consideramos separadamente os seguintes dois casos:

(i) $|L| \leq K$ $(L \notin Z_K)$;

(ii) $|L| > K$ $(L \in Z_K)$.

Lembre que, no caso (i), temos

$$c(L) \leq c^*(L), \qquad (3\text{-}20)$$

pois o ECBM sempre constrói uma ABB ótima para L quando $|L| \leq K$. Por isto, o principal limite superior apresentado nesta subseção, dado pelo Teorema 3.13, é função dos postos e das cardinalidades dos subvetores que pertencem a Z_K, e dos custos de ABBs ótimas para os subvetores que pertencem a $Z - Z_K$.

Agora, consideramos o caso (ii). O próximo teorema fornece um limite superior para $c(L)$, como função dos níveis dos nós da AEBB auxiliar construída pela Função Caso-Médio (ver subseção 3.4.1). Vale lembrar que

denotamos por T_L a ABB retornada pela Função Construir para o subvetor L.

Teorema 3.11 *Para todo $j \in I(L) \cup I'(L)$, seja l_j o nível do nó associado ao subvetor L_j na AEBB auxiliar T_D construída pela Função Caso-Médio($|L_1|, |L_2|, \ldots, |L_{2k+1}|$). Neste caso, temos*

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} (l_j + 1) \right) + \sum_{j \in I(L)} c(L_j)$$

Prova. Seja a uma chave do subvetor L_j, para um dado $j \in I'(L) \cup I(L)$. O nó a tem dois tipos de ancestral em T_L: aqueles que pertencem a L_j e aqueles que não pertencem a L_j. Por construção de T_D e T_L, existe uma relação de um para um entre os ancestrais de a em T_L que não pertencem a L_j e os ancestrais do nó associado a L_j em T_D. Por isto, o número de ancestrais de a em T_L que não pertencem a L_j é sempre igual a l_j. Além disso, qualquer ancestral de a em T_L que não pertence a L_j tem posto $r(L)$. Logo, o custo de buscar a segundo a árvore T_L não pode ser maior do que

$$\frac{l_j}{\alpha^{t-r(L)+1}} + \sum_{x \in (N_L(a) \cap L_j)} c(x),$$

onde denotamos por $N_L(a)$ o conjunto de todos os ancestrais do nó a em T_L.

Por construção de T_D e T_L, também existe uma relação de um para um entre os ancestrais de a em T_L que pertencem a L_j e os ancestrais de a em T_{L_j}. Por isto, somando o limite superior anterior para todas as chaves de L_j, obtemos o seguinte limite superior para o custo de buscar todas as chaves do subvetor L_j segundo a árvore T_L:

$$\sum_{a \in L_j} \left(\frac{l_j}{\alpha^{t-r(L)+1}} + \sum_{x \in (N_L(a) \cap L_j)} c(x) \right) = \frac{|L_j| l_j}{\alpha^{t-r(L)+1}} + c(L_j).$$

Como temos $c(L_j) \leq 1/\alpha^{t-r(L)+1}$ para todo $j \in I'(L)$, somando o limite superior anterior para todo $j \in I(L) \cup I'(L)$, obtemos que

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} l_j \right) + \sum_{j \in I'(L)} c(L_j) + \sum_{j \in I(L)} c(L_j)$$

$$\leq \frac{1}{\alpha^{t-r(L)+1}} \left(\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} l_j \right) + \frac{|I'(L)|}{\alpha^{t-r(L)+1}} + \sum_{j \in I(L)} c(L_j)$$

$$= \frac{1}{\alpha^{t-r(L)+1}} \left(\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} (l_j + 1) \right) + \sum_{j \in I(L)} c(L_j).$$

□

Agora, refinamos o resultado anterior, obtendo um limite superior para $c(L)$ onde os níveis dos nós da AEBB auxiliar T_D não aparecem. Para isto, utilizamos o fato de que a Função Caso-Médio constrói uma AEBB auxiliar recursivamente, utilizando o critério Min-Max para escolha da raiz (ver subseção 3.4.1).

Corolário 3.12

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i)$$

Prova. Sejam $h_1, \ldots, h_{2n+1}$ os níveis dos nós de uma AEBB construída recursivamente, utilizando o critério Min-Max para escolha da raiz. Além disso, sejam $w_1, \ldots, w_{2n+1}$ respectivos pesos. Neste caso, Bayer [6] prova que

$$\sum_{i=0}^{n} w_{2i+1} h_{2i+1} + \sum_{i=1}^{n} w_{2i} h_{2i} \leq \left(\sum_{i=1}^{2n+1} w_i \right) \log \left(\sum_{i=1}^{2n+1} w_i \right) - \sum_{i=1}^{2n+1} (w_i \log w_i) + 2 \sum_{i=1}^{2n+1} w_i.$$

De fato, o limite superior provado por Bayer é menor do que o limite dado pela desigualdade anterior. No entanto, não precisamos deste limite mais justo em nossas demonstrações. Também vale mencionar que a expressão deste limite superior que é apresentada em [9] é um diferente devido à normalização que é aplicada aos pesos.

Agora, considere o caso da AEBB auxiliar T_D construída pela Função Caso-Médio, para a lista de pesos $[|L_1|, \ldots, |L_{2k+1}|]$. Como $l_1, \ldots, l_{2k+1}$ são os níveis dos respectivos nós em T_D, pela desigualdade anterior, temos

$$\sum_{j \in I(L)} |L_j| l_j + \sum_{j \in I'(L)} (l_j + 1) \leq |L| \log |L| + 2|L| - \sum_{j \in I(L)} |L_j| \log |L_j|$$

Logo, combinando a desigualdade anterior com o Teorema 3.11 obtemos que

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i).$$

□

Agora, apresentamos a prova do principal limite superior para o custo da ABB construída pelo ECBM para a lista de entrada $\mathcal{A}$.

Teorema 3.13

$$c(\mathcal{A}) \leq \sum_{L \in Z_K} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z - Z_K)} \left(c^*(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right)$$

Prova. Demonstramos este teorema por indução sobre os níveis da árvore T(Construir). Para isto, utilizamos a mesma notação da prova do Teorema 3.10. Seja l um número natural. Neste caso, basta provar que

$$c(\mathcal{A}) \leq \sum_{L \in Z_K^l} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z^{l+1} - Z_K^l)} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right), \tag{3-21}$$

pois (3-21) equivale ao limite superior deste teorema quando fazemos l igual à altura da árvore T(Construir). Isto pode ser verificado observando-se que, neste caso, temos $Z_K^l = Z_K$, $Z^{l+1} = Z$ e $c(L) = c^*(L)$ para todo $L \in Z - Z_K$.

Provamos a corretude de (3-21) por indução sobre o valor de l. Seja $l = 0$. Se $|\mathcal{A}| \leq K$, então a desigualdade vale porque $Z^1 = \{\mathcal{A}\}$, $Z_K^0 = \emptyset$ e $g(\mathcal{A}) = -\infty$. Por isto, assumimos também que $|\mathcal{A}| > K$. Neste caso, observamos que $Z_K^0 = \{\mathcal{A}\}$ e definimos $\{\mathcal{A}_i | i \in I(\mathcal{A})\} = Z^1 - Z_K^0$. Com isto, precisamos provar que

$$c(\mathcal{A}) \leq \frac{1}{\alpha^{t-r(\mathcal{A})+1}} \left(|\mathcal{A}| \log |\mathcal{A}| + 2|\mathcal{A}| - \sum_{i \in I(\mathcal{A})} |\mathcal{A}_i| \log |\mathcal{A}_i| \right) + \sum_{i \in I(\mathcal{A})} c(\mathcal{A}_i).$$

Como a desigualdade anterior é uma conseqüência imediata do Corolário 3.12, para $L = \mathcal{A}$, temos que (3-21) vale para $l = 0$.

Agora, assumindo que esta desigualdade vale para $l = p$, provamos que ela também vale para $l = p + 1$. Neste caso, pela hipótese indutiva, temos que

$$c(\mathcal{A}) \leq \sum_{L \in Z_K^p} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z^{p+1} - Z_K^p)} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right). \tag{3-22}$$

De forma semelhante à prova do Teorema 3.10, o passo indutivo consiste em expandir as parcelas do lado direito da desigualdade anterior que correspondem aos subvetores que estão em nível $p+1$ em T(Construir) e têm mais de K chaves. Para isto, aplicamos o Corolário 3.12. Como o conjunto $Z_K^{p+1} - Z_K^p$ contém exatamente estes subvetores, reescrevemos o conjunto $Z^{p+1} - Z_K^p$ como a união de dois conjuntos disjuntos dados respectivamente por $Z^{p+1} - Z_K^{p+1}$ e $Z_K^{p+1} - Z_K^p$. Daí, concluímos que

$$c(\mathcal{A}) \leq \sum_{L \in Z_K^p} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \sum_{L \in (Z_K^{p+1} - Z_K^p)} c(L) - \sum_{L \in (Z_K^{p+1} - Z_K^p)} \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \sum_{L \in (Z^{p+1} - Z_K^{p+1})} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right). \tag{3-23}$$

Aplicando o Corolário 3.12 para um dado subvetor $L \in (Z_K^{p+1} - Z_K^p)$, obtemos que

$$c(L) \leq \frac{1}{\alpha^{t-r(L)+1}} \left(|L| \log |L| + 2|L| - \sum_{i \in I(L)} |L_i| \log |L_i| \right) + \sum_{i \in I(L)} c(L_i).$$

Somando a desigualdade anterior para todos os subvetores que pertencem a $(Z_K^{p+1} - Z_K^p)$, obtemos que

$$\sum_{L \in (Z_K^{p+1} - Z_K^p)} c(L) \leq \sum_{L \in (Z_K^{p+1} - Z_K^p)} \frac{|L| \log |L| + 2|L|}{\alpha^{t-r(L)+1}} - \sum_{L \in (Z^{p+2} - Z^{p+1})} \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \sum_{L \in (Z^{p+2} - Z^{p+1})} c(L). \tag{3-24}$$

Substituindo-se o segundo somatório do lado direito de (3-23) pelo lado direito de (3-24), obtemos que

$$\begin{aligned} c(\mathcal{A}) \leq & \sum_{L \in Z_K^p} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \\ & \sum_{L \in (Z_K^{p+1} - Z_K^p)} \left(\frac{|L| \log |L|}{\alpha^{t-r(L)+1}} - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}} \right) + \\ & \sum_{L \in (Z^{p+2} - Z^{p+1})} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right) + \\ & \sum_{L \in (Z^{p+1} - Z_K^{p+1})} \left(c(L) - \frac{|L| \log |L|}{\alpha^{t-g(L)+1}} \right). \end{aligned}$$

A corretude de (3-21) para $l = p+1$ é uma conseqüência imediata da desigualdade anterior e do fato de que $(Z^{p+2} - Z^{p+1})$ e $(Z^{p+1} - Z_K^{p+1})$ são dois conjuntos disjuntos cuja união é dada por $(Z^{p+2} - Z_K^{p+1})$. Isto completa esta prova. □

Fator de Aproximação

Nesta subseção, provamos que o Algoritmo ECBM sempre obtém soluções $(2 + \epsilon + o(1))$-aproximadas para o PBCM, para qualquer $\epsilon > 0$. Para isto, apresentamos uma comparação entre o limite inferior dado pelo Teorema 3.10 e o limite superior dado pelo Teorema 3.13. Como o valor de ϵ depende do valor de K utilizado na execução do algoritmo, mostramos que, para qualquer $\epsilon > 0$, sempre existe um valor de K que garante o fator de aproximação $(2 + \epsilon + o(1))$. Em nossas análises, sempre assumimos que $K \geq 2$.

Teorema 3.14

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \frac{n \log n + 2n}{\alpha^t} + \max \left\{ \frac{\alpha}{\alpha - 1}, \frac{\alpha(\alpha-1)\log(K+1) + 2\alpha^2}{(\alpha-1)(\log(K+1) - 1)} \right\}$$

Prova. Se $|\mathcal{A}| \leq K$, então o ECBM obtém uma ABB ótima. Neste caso, este teorema vale porque $c(\mathcal{A}) = c^*(\mathcal{A})$. Portanto, assumimos que $|\mathcal{A}| > K$. Dividimos nossa análise em dois casos:

(i) $r(\mathcal{A}) = 1$;

(ii) $r(\mathcal{A}) > 1$.

Primeiro, analisamos o caso (i). Pelo Teorema 3.13 temos que

$$c(\mathcal{A}) \leq \frac{|\mathcal{A}|(\log|\mathcal{A}|+2)}{\alpha^t} + \sum_{L\in(Z_K-\{\mathcal{A}\})}\left(\frac{|L|\log|L|}{\alpha^{t-r(L)+1}} - \frac{|L|\log|L|}{\alpha^{t-g(L)+1}} + \frac{2|L|}{\alpha^{t-r(L)+1}}\right) + \sum_{L\in(Z-Z_K)} c^*(L). \tag{3-25}$$

Por outro lado, o Teorema 3.10 garante que

$$c^*(\mathcal{A}) \geq \max\left\{1, \sum_{L\in Z_K}(R(L)-G(L))|L|(\log|L|-1) + \sum_{L\in(Z-Z_K)} c^*(L,G(L))\right\}. \tag{3-26}$$

Dividindo (3-25) pela desigualdade anterior e utilizando o fato de que $1/\alpha^{t-r(L)+1} = \alpha R(L)$ e $\alpha G(L) \leq 1/\alpha^{t-g(L)+1}$ para todo $L \in Z_K - \{\mathcal{A}\}$, obtemos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \frac{|\mathcal{A}|\log|\mathcal{A}|+2|\mathcal{A}|}{\alpha^t} + \frac{\sum_{L\in(Z_K-\mathcal{A})}(\alpha(R(L)-G(L))|L|\log|L|+2\alpha R(L)|L|) + \sum_{L\in(Z-Z_K)} c^*(L)}{\sum_{L\in Z_K}(R(L)-G(L))|L|(\log|L|-1) + \sum_{L\in(Z-Z_K)} c^*(L,G(L))}. \tag{3-27}$$

Agora, sejam

$$P = \max_{L\in(Z-Z_K)}\left\{\frac{c^*(L)}{c^*(L,G(L))}\right\}$$

e

$$Q = \max_{L\in(Z_K-\mathcal{A})}\left\{\frac{\alpha(R(L)-G(L))\log|L|+2\alpha R(L)}{(R(L)-G(L))(\log|L|-1)}\right\}.$$

A partir de (3-27), concluímos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \frac{|\mathcal{A}|\log|\mathcal{A}|+2|\mathcal{A}|}{\alpha^t} + \max\{P,Q\}. \tag{3-28}$$

Primeiro, analisamos P. Como temos $0 \leq G(L) < R(L) \leq \min_{a\in L}\{c(a)\}$ para todo subvetor $L \in (Z - Z_K)$, o Teorema 3.9 garante que

$$P = \max_{L \in (Z - Z_K)} \left\{ \frac{c^*(L)}{c^*(L, G(L))} \right\} \leq \max_{L \in (Z - Z_K)} \left\{ \frac{R(L)}{R(L) - G(L)} \right\}.$$

No entanto, sempre temos $G(L) \leq R(L)/\alpha$. Daí, concluímos que

$$P \leq \max_{L \in (Z - Z_K)} \left\{ \frac{R(L)}{R(L) - G(L)} \right\} \leq \frac{\alpha}{\alpha - 1}. \tag{3-29}$$

Agora, analisamos Q. Como $G(L) \leq R(L)/\alpha$ e $|L| > K$, para todo $L \in (Z_K - \mathcal{A})$, concluímos que

$$\begin{aligned} Q &\leq \max_{L \in (Z_K - \mathcal{A})} \left\{ \frac{\alpha(R(L) - G(L))}{(R(L) - G(L))(1 - 1/\log|L|)} + \right. \\ &\quad \left. \frac{2\alpha R(L)}{(R(L) - R(L)/\alpha)(\log|L| - 1)} \right\} \\ &\leq \frac{\alpha}{1 - 1/\log(K+1)} + \frac{2\alpha}{(1 - 1/\alpha)(\log(K+1) - 1)} \\ &= \frac{\alpha \log(K+1)}{\log(K+1) - 1} + \frac{2\alpha^2}{(\alpha - 1)(\log(K+1) - 1)}. \end{aligned} \tag{3-30}$$

Como $|\mathcal{A}| = n$, combinando (3-28), (3-29) e (3-30), obtemos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \frac{n \log n + 2n}{\alpha^t} + \max \left\{ \frac{\alpha}{\alpha - 1}, \frac{\alpha(\alpha - 1)\log(K+1) + 2\alpha^2}{(\alpha - 1)(\log(K+1) - 1)} \right\}. \tag{3-31}$$

Finalmente, analisamos o caso (ii), onde $r(\mathcal{A}) > 1$. Este caso é muito semelhante ao caso (i). A única diferença é o fato de que o vetor de $\mathcal{A}$ não precisa ser tratado separadamente. Logo, pelo Teorema 3.13, temos que

$$c(\mathcal{A}) \leq \sum_{L \in Z_K} \left(\frac{|L| \log|L|}{\alpha^{t - r(L) + 1}} - \frac{|L| \log|L|}{\alpha^{t - g(L) + 1}} + \frac{2|L|}{\alpha^{t - r(L) + 1}} \right) + \sum_{L \in (Z - Z_K)} c^*(L). \tag{3-32}$$

Dividindo (3-32) por (3-26) e utilizando o fato de que $1/\alpha^{t - r(L) + 1} = \alpha R(L)$ e $\alpha G(L) \leq 1/\alpha^{t - g(L) + 1}$ para todo $L \in Z_K$, obtemos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq$$

$$\frac{\sum_{L\in Z_K}(\alpha(R(L)-G(L))|L|\log|L|+2\alpha R(L)|L|)+\sum_{L\in(Z-Z_K)}c^*(L)}{\sum_{L\in Z_K}(R(L)-G(L))|L|(\log|L|-1)+\sum_{L\in(Z-Z_K)}c^*(L,G(L))}. \tag{3-33}$$

Seja

$$Q' = \max_{L\in Z_K}\left\{\frac{\alpha(R(L)-G(L))\log|L|+2\alpha R(L)}{(R(L)-G(L))(\log|L|-1)}\right\}.$$

A partir de (3-33), concluímos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \max\{P, Q'\}. \tag{3-34}$$

Como $G(L) \leq R(L)/\alpha$ e $|L| > K$, para todo $L \in Z_K$, concluímos que

$$Q' \leq \frac{\alpha\log(K+1)}{\log(K+1)-1} + \frac{2\alpha^2}{(\alpha-1)(\log(K+1)-1)}. \tag{3-35}$$

Vale mencionar que a análise de Q' é idêntica à análise que resulta em (3-30). Combinando (3-34), (3-29) e (3-35), obtemos que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \max\left\{\frac{\alpha}{\alpha-1}, \frac{\alpha(\alpha-1)\log(K+1)+2\alpha^2}{(\alpha-1)(\log(K+1)-1)}\right\}, \tag{3-36}$$

o que conclui esta prova. □

Corolário 3.15 *O Algoritmo ECBM é* $(2+\epsilon+o(1))$*-aproximado, para qualquer* $\epsilon > 0$.

Prova. Considere um valor arbitrário de $\epsilon > 0$. Fazendo $\alpha = 2$, $t = 2\log n$ e $K = 2^{1+\lceil 10/\epsilon\rceil} - 1$, obtemos a partir do Teorema 3.14 que

$$\frac{c(\mathcal{A})}{c^*(\mathcal{A})} \leq \frac{n\log n+2n}{n^2} + 2 + \frac{10}{\lceil 10/\epsilon\rceil} \leq \frac{n\log n+2n}{n^2} + 2 + \epsilon.$$

□

3.5.5
Análise do Algoritmo ECPB

Nesta subseção, analisamos o fator de aproximação do Algoritmo ECPB. Para isto, utilizamos os resultados e a notação definida na subseção 3.5.3. Além disso, definimos uma árvore $T_{pv}(L)$ onde cada nó está associado ao pivô de um subvetor L_i na árvore $T^*_W(L)$, para $i \in I'(L) \cup I(L)$. Para cada pivô y_i, seja $\phi(y_i)$ o ancestral estrito de y_i que tem o maior nível e que é pivô de algum subvetor L_j em $T^*_W(L)$, para $j \in I'(L) \cup I(L)$. A árvore $T_{pv}(L)$ pode ser construída da seguinte forma.

Para todo $y \in \{y_i | i \in I'(L) \cup I(L)\}$ **faça**
  Se y é descendente do filho à esquerda de $\phi(y)$ em $T^*_W(L)$ **então**
    conectar y como filho à esquerda de $\phi(y)$ em $T_{pv}(L)$;
  Senão
    conectar y como filho à direita de $\phi(y)$ em $T_{pv}(L)$;
  Fim-Se
Fim-Para

A Figura 3.5.(b) mostra um exemplo de uma árvore $T_{pv}(L)$ construída a partir da ABB $T^*_W(L)$ representada pela Figura 3.5.(a), para o caso em que o subvetor $L = [a_1, \ldots, a_9]$ é particionado nos subvetores $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$ e $L_7 = []$.

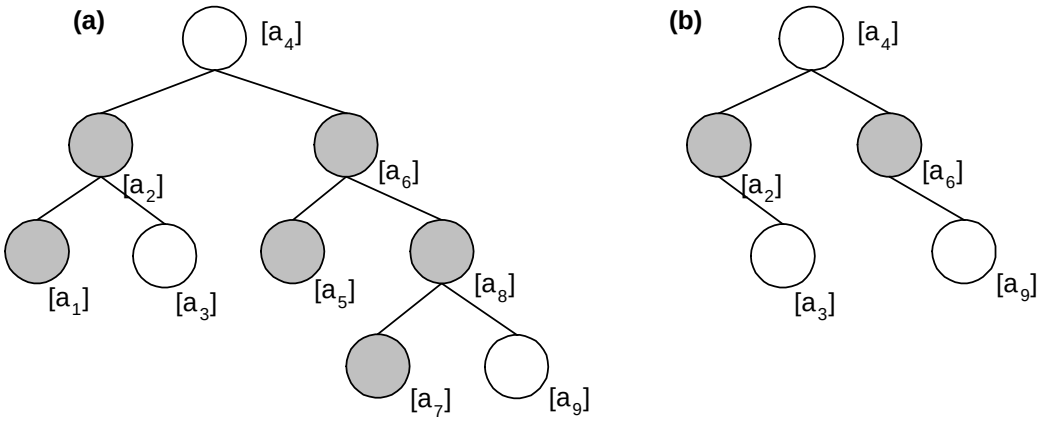

Figura 3.5: (a) Uma ABB ótima $T^*_W(L)$ para o subvetor $L = [a_1, \ldots, a_9]$. (b) A árvore $T_{pv}(L)$ correspondente, para a partição de L em $L_1 = [a_1, a_2]$, $L_2 = [a_3]$, $L_3 = []$, $L_4 = [a_4]$, $L_5 = [a_5, \ldots, a_8]$, $L_6 = [a_9]$ e $L_7 = []$.

Neste caso, temos a seguinte proposição.

Proposição 3.16 *$T_{pv}(L)$ é uma ABB.*

Prova. Seja y_r a raiz de $T^*_W(L)$. Além disso, seja r tal que L_r contém a chave y_r. Pela definição de pivô, y_r é o pivô de L_r. Daí, podemos concluir que qualquer pivô $y_i \neq y_r$ de um subvetor L_i, para $i \in I'(L) \cup I(L)$, tem pelo menos um ancestral estrito em $T^*_W(L)$ que é pivô de um subvetor L_j, para $j \in I'(L) \cup I(L)$. Logo, qualquer $y_i \neq y_r$ tem um pai em $T_{pv}(L)$.

Agora, considere uma seqüência de nós construída da seguinte forma. Começamos com um nó qualquer y_i de $T_{pv}(L)$. Se a é o nó atual, então o próximo nó é o pai de a em $T_{pv}(L)$. Se tal pai não existe, então a seqüência termina. Por construção de $T_{pv}(L)$, esta seqüência não pode ter nós repetidos. Portanto, ela termina. Como nó de $T_{pv}(L)$ exceto y_r tem um pai em $T_{pv}(L)$, esta seqüência termina em y_r independentemente do nó de início. Daí concluímos que $T_{pv}(L)$ tem uma única raiz y_r, que é ancestral de todos os seus nós.

Com isto, basta provar que qualquer nó y_i tem no máximo um filho à esquerda e um à direita, em $T_{pv}(L)$. Provamos isto por contradição apenas para o filho à esquerda, pois o caso do filho à direita é simétrico. Sejam y_j e y_k dois filhos à esquerda de y_i em $T_{pv}(L)$. Assumimos sem perder generalidade que a chave y_j está à esquerda da chave y_k no subvetor L. Observe que y_j não é ancestral nem descendente de y_k em $T^*_W(L)$. Em caso contrário, y_i não poderia ser escolhido como pai de ambos os nós y_j e y_k. Neste caso, como $T^*_W(L)$ é uma ABB, y_j e y_k têm pelo menos um ancestral comum a cuja chave está entre y_j e y_k no subvetor L. Além disso, como a também está entre y_j e y_i e y_i é ancestral de y_j, y_i também é ancestral do nó a em $T^*_W(L)$. Pelo mesmo argumento, podemos concluir que a é ancestral de qualquer nó de $T^*_W(L)$ cuja chave está entre y_j e y_k. Como y_j e y_k são nós de $T_{pv}(L)$, eles são pivôs seus respectivos subvetores L_j e L_k. Logo, a pertence a outro subvetor, que está entre L_j e L_k. Em caso contrário, a seria o pivô de L_j ou de L_k. Como a é ancestral de qualquer nó de $T^*_W(L)$ cuja chave está entre y_j e y_k, concluímos que a é pivô do subvetor a que ele pertence. Por isto, existe um nó associado a a em $T_{pv}(L)$. Como a é descendente de y_i e ancestral de y_j em $T^*_W(L)$, y_i não poderia ter sido escolhido como pai de y_j. Daí, concluímos que y_i tem no máximo um filho à esquerda em $T_{pv}(L)$. □

Nossa estratégia para demonstrar um fator de aproximação constante para o ECPB é obter um limite inferior para o valor de $w^*(L)$ e um limite superior para o valor de $w(L)$, ambos como função dos níveis dos nós de $T_{pv}(L)$. Desta forma, é possível analisar a razão entre estes dois limites.

Para isto, utilizamos uma propriedade de ABBs para relacionar os níveis das folhas da AEBB auxiliar T_D com os níveis dos nós de $T_{pv}(L)$. Esta propriedade é apresentada no próximo lema.

Lema 6 *Para $i \in I(L) \cup I'(L)$, seja l_i o nível de y_i em $T_{pv}(L)$. Neste caso, existe uma ABB T' com os mesmos nós de $T_{pv}(L)$, onde:*

(i) para cada $i \in I(L)$, y_i é uma folha de T' em nível menor ou igual a $2l_i + 2$;

(ii) para cada $i \in I'(L)$, y_i é um nó (folha ou nó interno) de T' em nível menor ou igual a $2l_i$.

Prova. Provamos este lema por indução sobre o número de nós de $T_{pv}(L)$.

Para $|T_{pv}(L)| = 1$, temos $I(L) = \{1\}$, $I'(L) =$ e $l_1 = 0$. Como y_1 é uma folha em $T_{pv}(L)$, fazemos $T' = T_{pv}(L)$, o que satisfaz este lema.

Agora, assumimos que este lema vale para qualquer ABB $T_{pv}(L)$ com menos do que m nós. Neste caso, dada uma ABB $T_{pv}(L)$ com exatamente m nós, temos que provar que o lema também vale para $T_{pv}(L)$.

Seja y_r a raiz de $T_{pv}(L)$. Denotamos por T^ℓ e T^r as sub-árvores à esquerda e à direita de y_r em $T_{pv}(L)$, respectivamente. Além disso, sejam $\hat{T}^\ell$ e $\hat{T}^r$ as duas árvores obtidas a partir de T^ℓ e T^r, respectivamente, aplicando a hipótese indutiva. Vale observar que o nível de qualquer nó y_i, para $i \neq r$, em T^ℓ ou T^r é $l_i - 1$.

Dividimos em dois casos:

(a) $r \in I'(L)$;

(b) $r \in I(L)$.

No caso (a), podemos construir uma ABB T' da seguinte forma: fazemos de y_r a raiz T' e de $\hat{T}^\ell$ e $\hat{T}^r$ as sub-árvores à esquerda e à direita de y_r, respectivamente. Por construção, o nível de y_i em T', para qualquer $i \in I(L)$, não é maior do que $2(l_i - 1) + 2 + 1 < 2l_i + 2$, o que satisfaz a condição (i). Além disso, o nível de y_i em T', para qualquer $i \in (I'(L) - \{r\})$, não é maior do que $2(l_i - 1) + 1 < 2l_i$, o que satisfaz a condição (ii).

Para o caso (b), assumimos inicialmente que $2 < r < 2k$. Além disso, utilizamos dois nós temporários y e y'. Com isto, podemos construir uma ABB T' da seguinte forma:

1. fazemos de y a raiz de T';
2. fazemos de y' o filho à esquerda de y;
3. fazemos de y_r o filho à direita de y';
4. fazemos de $\hat{T}^\ell$ a sub-árvore à esquerda de y'

5. fazemos de $\hat{T}^r$ a sub-árvore à direita de y.

A Figura 3.6 representa a construção descrita anteriormente. As árvores $T_{pv}(L)$ e T' são representadas respectivamente pelas Figuras 3.6.(a) e 3.6.(b).

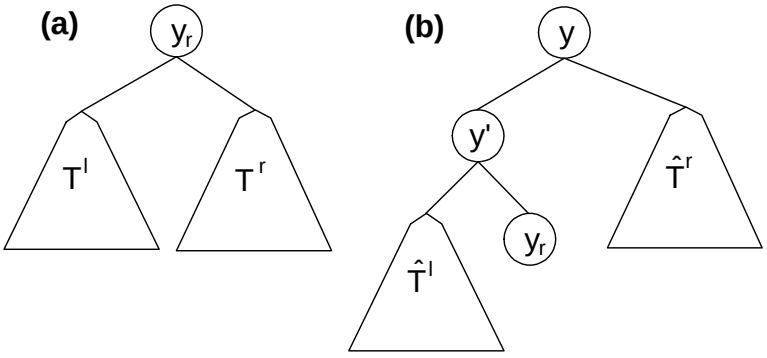

Figura 3.6: (a) Uma ABB $T_{pv}(L)$ de pivôs onde $r \in I(L)$. (b) Uma ABB T' obtida a partir de $T_{pv}(L)$, satisfazendo as condições do lema 6.

Observe que o nó y_r é uma folha em nível 2 em T'. Além disso, por construção, o nível de y_i em T', para qualquer $i \in I(L) - \{r\}$, não é maior do que $2(l_i - 1) + 2 + 2 = 2l_i + 2$. Adicionalmente, o nível de y_i em T', para qualquer $i \in I'(L)$, não é maior do que $2(l_i - 1) + 2 = 2l_i$. Portanto, as duas condições são satisfeitas no que diz respeito aos níveis dos nós em T'. Agora, basta substituir os nós temporários y' e y em T' por nós removidos de T^ℓ e T^r, respectivamente, respeitando a ordem definida pelo subvetor L. Neste caso, os nós y' e y devem ser substituídos respectivamente pelo nó mais à direita y_{r-1} de $\hat{T}^\ell$ e pelo nó mais à esquerda y_{r+1} de $\hat{T}^r$. Quando removemos y_{r-1} (y_{r+1}) de T^ℓ (T^r), conectamos o filho do nó removido, se existir, ao nó que era pai do nó removido. Observe que nenhuma destas operações aumenta o nível de nenhum nó de T'. Por isto, como $r-1$ e $r+1$ pertencem $I'(L)$, as condições (i) e (ii) valem para T'.

Finalmente, basta analisar o caso (b) para $r = 1$, pois o caso (b) para $r = 2k + 1$ é simétrico. Neste caso, a construção de T' é idêntica à construção anterior, exceto pela escolha do nó que substitui y'. Como T^ℓ não existe neste caso, substituímos y' por y_r. Com isto, como y_r continua sendo uma folha de T' e as considerações feitas sobre os níveis dos nós em T' continuam valendo, as condições (i) e (ii) valem para T'. □

Limites Inferiores

Nesta subseção, apresentamos limites inferiores para o valor de $w^*(L)$. O próximo lema apresenta o principal limite inferior desta subseção, que é uma aplicação direta do Lema 5.

Lema 7 *Seja l_i o nível de y_i em $T_{pv}(L)$. Se $r(L) = 1$, então*

$$w^*(L) \geq \max_{i \in I(L) \cup I'(L)} \{w^*(L_i)\}. \tag{3-37}$$

Por outro lado, se $r(L) > 1$, então

$$w^*(L) \geq \max_{i \in I(L) \cup I'(L)} \{l_i/\alpha^{t-r(L)+2} + w^*(L_i)\}. \tag{3-38}$$

Prova. Seja l_i^W o nível de y_i em $T_W^*(L)$. Por construção de $T_{pv}(L)$, sempre temos $l_i \leq l_i^W$, para todo $i \in I(L) \cup I'(L)$. Logo, (3-37) e (3-38) são conseqüências imediatas do Lema 5 para $d = 0$ e para $d = 1/\alpha^{t-r(L)+2}$, respectivamente. □

Em seguida, demonstramos mais um limite inferior para $w^*(L)$, que será utilizado apenas para analisar o impacto dos subvetores com K chaves ou menos sobre o fator de aproximação do ECPB. Este limite inferior é necessário porque o limite dado pelo Lema 7 só pode ser aplicado aos subvetores com mais de K chaves, que são particionados pela Função Construir.

Proposição 3.17 *Seja L um subvetor de $\mathcal{A}$, com $r(L) > 1$. Neste caso,*

$$w^*(L) \geq \frac{\log(|L|+1)}{\alpha^{t-r(L)+2}}$$

Prova. Observe que $w^*(L)$ não pode ser menor do que o pior custo de busca de uma ABB ótima T^* para um vetor de $|L|$ chaves, onde o custo de acesso de qualquer chave é $1/\alpha^{t-r(L)+2}$. Logo, esta proposição é uma conseqüência imediata da Proposição 3.2. □

Limites Superiores

Nesta subseção, apresentamos limites superiores para o valor de $w(L)$. O primeiro limite superior, dado pelo próximo lema, é função dos níveis das folhas da AEBB auxiliar T_D construída pela Função Pior-Caso.

Lema 8 *Seja T_D uma AEBB auxiliar T_D construída pela Função Pior-Caso, para os argumentos de entrada $w(L_1), w(L_3), \ldots, w(L_{2k+1})$. Neste caso, temos*

$$w(L) \leq w'(T_D) = \max_{i=0,\ldots,k} \left\{ l^D_{2i+1}/\alpha^{t-r(L)+1} + w(L_{2i+1}) \right\},$$

onde $l^D_1, l^D_3, \ldots, l^D_{2k+1}$ são os respectivos níveis das folhas de T_D e a função $w'()$ é definida por (3-9).

Prova. Seja a uma chave de L cujo custo de busca segundo T_L é máximo. Denotamos por $N_L(a)$ o conjunto de todos os ancestrais de a em T_L. Além disso, seja $i \in I(L) \cup I'(L)$ tal que $a \in L_i$.

Se $i \in I(L)$, então concluímos que

$$w(L) = \sum_{x \in N_L(a) \cap L_i} c(x) + \sum_{x \in N_L(a) - L_i} c(x) = w(L_i) + \sum_{x \in N_L(a) - L_i} c(x).$$

Neste caso, a segunda igualdade se deve ao fato de que a chave a também tem um custo de busca máximo segundo T_{L_i}. Além disso, como o custo de acesso de qualquer nó interno de T_D é menor ou igual a $1/\alpha^{t-r(L)+1}$, obtemos que

$$w(L) = w(L_i) + \sum_{x \in N_L(a) - L_i} c(x) \leq w(L_i) + l^D_i/\alpha^{t-r(L)+1} \leq w'(T_D).$$

Por outro lado, se $i \in I'(L)$, então $i+1 \in I(L)$. Além disso, como l^D_i é o nível de um nó interno, l^D_{i+1} é o nível de uma folha e os dois têm índices consecutivos em T_D, concluímos que $l^D_i \leq l^D_{i+1}$. Logo,

$$w(L) = \sum_{x \in N_L(a)} c(x) \leq l^D_{i+1}/\alpha^{t-r(L)+1} \leq w'(T_D)$$

□

Agora, apresentamos o principal limite superior desta subseção. Assim como o limite inferior dado pelo Lema 7, este limite superior é função

dos níveis dos nós de $T_{pv}(L)$. Vale observar que, exceto por alguns termos aditivos, os dois limites diferem por um fator de 2α.

Para provar o próximo limite superior, construímos um AEBB T'' acrescentando folhas à ABB T' dada pelo Lema 6. Então, demonstramos que $w'(T_D) \leq w'(T'')$.

Lema 9 *Seja l_i o nível de y_i em $T_{pv}(L)$. Se $r(L) > 1$, então temos*

$$w(L) \leq \max_{i \in I(L) \cup I'(L)} \left\{ \frac{2l_i + 2}{\alpha^{t-r(L)+1}} + w(L_i) \right\} \tag{3-39}$$

Prova. Para simplificar a nossa notação, definimos $\gamma(L) = 1/\alpha^{t-r(L)+1}$. Seja T' a ABB dada pelo Lema 6. Além disso, denotamos por l'_i o nível de y_i em T', para qualquer $i \in I(L) \cup I'(L)$. Para $i \in \{1, 3, \ldots, 2k+1\} - I(L)$, definimos

$$l'_i = \begin{cases} l'_2 + 1, \text{ caso } i = 1 \\ \max\{l'_{i-1}, l'_{i+1}\} + 1, \text{ caso } 1 < i < 2k+1 \\ l'_{2k} + 1, \text{ caso } i = 2k+1 \end{cases}$$

Observe que podemos inserir na árvore T' uma folha em nível l'_i para cada valor de l'_i dado pela definição anterior. Neste caso, cada folha acrescentada corresponde a um subvetor vazio que pertence ao conjunto $\{L_i | i \in \{1, 3, \ldots, 2k+1\} - I(L)\}$. Seja T'' a AEBB resultante. Com isto, concluímos que

$$w'(T'') = \max_{i=1,3,\ldots,2k+1} \{l'_i \gamma(L) + w(L_i)\}. \tag{3-40}$$

Agora, seja

$$i^* = \arg\max_{i=1,3,\ldots,2k+1} \{l'_i \gamma(L) + w(L_i)\}.$$

Se $i^* \in I(L)$, então, pelo Lema 6, temos $l'_{i^*} \leq 2l_{i^*} + 2$. Logo, concluímos que

$$l'_{i^*} \gamma(L) + w(L_{i^*}) \leq (2l_{i^*} + 2)\gamma(L) + w(L_{i^*}).$$

Por outro lado, se $i^* \in \{1, 3, \ldots, 2k+1\} - I(L)$, então temos $l'_{i^*} = l'_j + 1$, para algum $j \in I'(L)$. Portanto, pelo Lema 6, temos $l'_{i^*} \leq 2l_j + 1$. Conseqüentemente,

$$l'_{i^*} \gamma(L) + w(L_{i^*}) = l'_{i^*} \gamma(L) \leq (2l_j + 2)\gamma(L) + w(L_j).$$

Daí, concluímos que

$$w'(T'') \leq \max_{i \in I(L) \cup I'(L)} \{(2l_i + 2)\gamma(L) + w(L_i)\}. \quad (3\text{-}41)$$

Como $r(L) > 1$, a Função Pior-Caso retorna uma AEBB T_D ótima para o Problema de Busca com Folgas, definido na subseção 3.4.2. Por isto, como T'' é uma solução viável para este mesmo problema, temos que $w'(T_D) \leq w'(T'')$. Logo, combinando (3-41) com a desigualdade dada pelo Lema 8, concluímos que

$$w(L) \leq w'(T_D) \leq w'(T'') \leq \max_{i \in I(L) \cup I'(L)} \{(2l_i + 2)\gamma(L) + w(L_i)\}.$$

□

Fator de Aproximação

Nesta subseção, provamos que o Algoritmo ECPB sempre obtém soluções $(2+\epsilon+o(1))$-aproximadas para o PBPC, para qualquer $\epsilon > 0$. Para isto, o próximo lema relaciona $w(\mathcal{A})$ e $w^*(\mathcal{A})$ no caso em que $r(\mathcal{A}) = 1$. Para qualquer subvetor L de $\mathcal{A}$ (inclusive o próprio) com $r(L) > 1$, a relação entre $w(L)$ e $w^*(L)$ é analisada pelo Lema 11. Com isto, provamos o fator de aproximação mencionado anteriormente no Teorema 3.18, aplicando sucessivamente os Lemas 10 e 11.

Lema 10 *Suponha que o vetor de entrada $\mathcal{A}$ é particionado pela Função Construir nos subvetores $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{2k+1}$.*

Além disso, seja m tal que $w(\mathcal{A}_m) = \max\{w(\mathcal{A}_i) | i \in \{1, 2, \ldots, 2k+1\}\}$. Se $r(\mathcal{A}) = 1$, então

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{n \log n + n}{\alpha^t} + \frac{w(\mathcal{A}_m)}{w^*(\mathcal{A}_m)}.$$

Prova. Como $r(\mathcal{A}) = 1$, a Função Pior-Caso retorna uma AEBB balanceada. Logo, temos

$$w(\mathcal{A}) \leq \frac{\lceil \log(k+1) \rceil}{\alpha^t} + w(\mathcal{A}_m) \leq \frac{\log n + 1}{\alpha^t} + w(\mathcal{A}_m).$$

Como $w^*(\mathcal{A}) \geq w^*(\mathcal{A}_m) > 0$ e $w^*(\mathcal{A}) \geq \max_{i=1,\ldots,n} \{c(a_i)\} \geq 1/n$, concluímos que

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{(\log n + 1)/\alpha^t}{1/n} + \frac{w(\mathcal{A}_m)}{w^*(\mathcal{A}_m)} = \frac{n \log n + n}{\alpha^t} + \frac{w(\mathcal{A}_m)}{w^*(\mathcal{A}_m)}.$$

□

O próximo lema compara o limite superior dado pelo Lema 9 como os limites inferiores dados pelo Lema 7 e pela Proposição 3.17.

Lema 11 *Seja L um subvetor passado como argumento para a Função Construir, com mais de K chaves e posto maior do que 1. Além disso, seja p um índice tal que o valor de*

$$\frac{2l_p + 2}{\alpha^{t-r(L)+1}} + w(L_p)$$

é máximo em $I(L) \cup I'(L)$. Se $|L_p| > K$, então

$$\frac{w(L)}{w^*(L)} \leq \max\left\{2\alpha, \frac{2/\alpha^{t-r(L)+1} + w(L_p)}{w^*(L_p)}\right\} \tag{3-42}$$

Em caso contrário, se $|L_p| \leq K$, então

$$\frac{w(L)}{w^*(L)} \leq \frac{2\alpha}{\log(K+1)} + 2\alpha \tag{3-43}$$

Prova. Primeiro, consideramos o caso em que $|L_p| > K$. Neste caso, pelo Lema 9, temos que $w(L) \leq (2l_p + 2)/\alpha^{t-r(L)+1} + w(L_p)$. Além disso, o Lema 7 garante que $w^*(L) \geq l_p/\alpha^{t-r(L)+2} + w^*(L_p)$. Dividindo a primeira desigualdade pela segunda, obtemos que

$$\begin{aligned}
\frac{w(L)}{w^*(L)} &\leq \frac{2l_p/\alpha^{t-r(L)+1} + 2/\alpha^{t-r(L)+1} + w(L_p)}{l_p/\alpha^{t-r(L)+2} + w^*(L_p)} \\
&\leq \max\left\{\frac{2l_p/\alpha^{t-r(L)+1}}{l_p/\alpha^{t-r(L)+2}}, \frac{2/\alpha^{t-r(L)+1} + w(L_p)}{w^*(L_p)}\right\} \\
&= \max\left\{2\alpha, \frac{2/\alpha^{t-r(L)+1} + w(L_p)}{w^*(L_p)}\right\}.
\end{aligned}$$

Agora, consideramos o caso em que $|L_p| \leq K$. Este caso é idêntico ao anterior, exceto pelo fato de que $w(L_p) = w^*(L_p)$, pois a Função Construir retorna uma ABB ótima para L_p. Logo, obtemos que

$$\frac{w(L)}{w^*(L)} \leq \frac{2/\alpha^{t-r(L)+1}}{w^*(L)} + \frac{2l_p/\alpha^{t-r(L)+1} + w^*(L_p)}{l_p/\alpha^{t-r(L)+2} + w^*(L_p)} \leq \frac{2/\alpha^{t-r(L)+1}}{w^*(L)} + 2\alpha. \tag{3-44}$$

Como, pela Proposição 3.17, temos $w^*(L) \geq \log(K+1)/\alpha^{t-r(L)+2}$, concluímos que

$$\frac{w(L)}{w^*(L)} \leq \frac{2\alpha}{\log(K+1)} + 2\alpha.$$

□

Finalmente, provamos um fator de aproximação para o ECPB.

Teorema 3.18

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{n\log n + n}{\alpha^t} + \frac{2\alpha^2}{(\alpha-1)\log(K+1)} + 2\alpha.$$

Prova. Se $|\mathcal{A}| \leq K$, então $w(\mathcal{A})/w^*(\mathcal{A}) = 1$. Por isto, basta considerar o caso em que $|\mathcal{A}| > K$. Neste caso, obtemos um limite superior para o valor de $w(\mathcal{A})/w^*(\mathcal{A})$ construtivamente da seguinte forma:

Passo 0: Começamos com a própria razão $w(\mathcal{A})/w^*(\mathcal{A})$ como o limite superior atual.

Passo 1: Se $r(\mathcal{A}) = 1$, então substituir esta razão pelo limite superior dado pelo Lema 10.

Passo 2: Enquanto o limite superior atual é função da razão $w(L)/w^*(L)$ para algum subvetor L de $\mathcal{A}$, substituir a expressão $w(L)/w^*(L)$ pelo limite superior dado pelo Lema 11.

Se $r(\mathcal{A}) = 1$, então, pelo Lema 10, temos que

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{n\log n + n}{\alpha^t} + \frac{w(\mathcal{A}_m)}{w^*(\mathcal{A}_m)}.$$

Neste caso, definimos $L^0 = \mathcal{A}_m$. Com isto, temos

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{n\log n + n}{\alpha^t} + \frac{w(L^0)}{w^*(L^0)}.$$

ao final do Passo 1. Em caso contrário, definimos $L^0 = \mathcal{A}$, o que nos dá

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} = \frac{w(L^0)}{w^*(L^0)}$$

ao final do Passo 1. Em ambos os casos, basta provar que o Passo 2 substitui $w(L^0)/w^*(L^0)$ por uma expressão menor ou igual a

$$\frac{2\alpha^2}{(\alpha-1)\log(K+1)} + 2\alpha. \tag{3-45}$$

Seja $g+1$ o número de iterações realizadas pelo Passo 2 e L^{i-1} o subvetor considerado como L na i-ésima iteração, para $i = 1, 2, \ldots, g+1$. Se $g = 0$, então o teorema vale, pois

$$\frac{2\alpha}{\log(K+1)} + 2\alpha < \frac{2\alpha^2}{(\alpha-1)\log(K+1)} + 2\alpha$$

Por isto, assumimos que $g > 0$. Neste caso, observe que o limite superior dado pelo Lema 11 é função da razão $w(L_p)/w^*(L_p)$ para algum subvetor L_p de L quando $|L_p| > K$. Em caso contrário, este limite não depende da razão $w(L_p)/w^*(L_p)$ para nenhum subvetor L_p de L. Daí, concluímos que $L^{i-1} > K$, para $i = 1, 2, \ldots, g+1$. Além disso, se o limite superior obtido após a i-ésima iteração do Passo 2 substitui $w(L^0)/w^*(L^0)$ por

$$\bar{w} + \max\left\{2\alpha, \frac{2/\alpha^{t-r(L^{i-1})+1} + w(L^i)}{w^*(L^i)}\right\},$$

para $i = 1, 2, \ldots, g-1$, então ele substitui $w(L^0)/w^*(L^0)$ por

$$\bar{w} + \max\left\{2\alpha, \frac{2/\alpha^{t-r(L^{i-1})+1}}{w^*(L^i)} + \max\left\{2\alpha, \frac{2/\alpha^{t-r(L^i)+1} + w(L^{i+1})}{w^*(L^{i+1})}\right\}\right\}$$

após a $(i+1)$-ésima iteração, o que pode ser reescrito como

$$\left(\bar{w} + \frac{2/\alpha^{t-r(L^{i-1})+1}}{w^*(L^i)}\right) + \max\left\{2\alpha, \frac{2/\alpha^{t-r(L^i)+1} + w(L^{i+1})}{w^*(L^{i+1})}\right\}.$$

Como $\bar{w} = 0$ após a primeira iteração do Passo 2, concluímos que o limite superior obtido após a última iteração do Passo 2 substitui $w(L^0)/w^*(L^0)$ por

$$\begin{aligned}
&\sum_{i=1}^{g-1} \frac{2/\alpha^{t-r(L^{i-1})+1}}{w^*(L^i)} + \max\left\{2\alpha, \frac{2/\alpha^{t-r(L^{g-1})+1}}{w^*(L^g)} + \frac{2\alpha}{\log(K+1)} + 2\alpha\right\} \\
&= \sum_{i=1}^{g} \frac{2/\alpha^{t-r(L^{i-1})+1}}{w^*(L^i)} + \frac{2\alpha}{\log(K+1)} + 2\alpha
\end{aligned} \tag{3-46}$$

Agora, basta provar que (3-46) não é maior do que (3-45). Para isto, observamos que

$$\begin{aligned}\sum_{i=1}^{g} \frac{2}{\alpha^{t-r(L^{i-1})+1}} &\leq \frac{2}{\alpha^{t-r(L^{g-1})+1}} \sum_{i=0}^{\infty} \frac{1}{\alpha^i} \\ &= \frac{2}{\alpha^{t-r(L^{g-1})+1}} \times \frac{1}{1-1/\alpha} \\ &\leq \frac{2/\alpha^{t-r(L^g)+1}}{\alpha-1}. \end{aligned} \tag{3-47}$$

Além disso, utilizando o fato de que $|L^g| > K$ e a Proposição 3.17, obtemos que

$$w^*(L^i) \geq w^*(L^g) \geq \frac{\log(K+1)}{\alpha^{t-r(L^g)+2}}, \tag{3-48}$$

para $i = 1, 2, \ldots, g$. Utilizando os limites dados por (3-47) e (3-48), concluímos que (3-46) não é maior do que

$$\frac{\alpha^{t-r(L^g)+2}}{\log(K+1)} \times \frac{2/\alpha^{t-r(L^g)+1}}{\alpha-1} + \frac{2\alpha}{\log(K+1)} + 2\alpha$$

$$= \frac{2\alpha}{(\alpha-1)\log(K+1)} + \frac{2\alpha}{\log(K+1)} + 2\alpha$$

$$= \frac{2\alpha^2}{(\alpha-1)\log(K+1)} + 2\alpha.$$

□

Corolário 3.19 *O Algoritmo ECPB é* $(2+\epsilon+o(1))$*-aproximado, para qualquer* $\epsilon > 0$.

Prova. Considere um valor arbitrário de $\epsilon > 0$. Fazendo $\alpha = 1 + \epsilon/4$, $t = \lceil 2\log_\alpha n \rceil$ e $K = 2^{\lceil 16\alpha^2/\epsilon^2 \rceil} - 1$, o Teorema 3.18 garante que

$$\frac{w(\mathcal{A})}{w^*(\mathcal{A})} \leq \frac{n\log n + n}{n^2} + \frac{8\alpha^2}{\lceil 16\alpha^2/\epsilon^2 \rceil \epsilon} + \epsilon/2 + 2 \leq \frac{n\log n + n}{n^2} + \epsilon + 2.$$

□

3.6 Implementação de Escalas de Custos

Nesta seção, apresentamos uma implementação dos algoritmos ECBM e ECPB, apresentados na seção 3.4. Esta implementação roda em tempo $O(n)$ e utiliza um espaço $O(n)$. Para isto, assumimos um modelo de computação onde operações de soma, subtração, multiplicação, divisão e potências de dois são executadas em tempo $O(1)$. Apesar desta consideração ser menos comum para a última operação, trata-se de uma hipótese razoável uma vez que calcular uma potência de dois corresponde deslocar os *bits* do operando, o que pode ser feito com apenas uma instrução de máquina na maioria dos computadores atuais. Por isto, consideramos inicialmente que $\alpha = 2$ e $t = \lceil \log n^2 \rceil$.

3.6.1 Cálculo dos Postos das Chaves

Seja $r(a_i)$ o posto da chave a_i. Calculamos os postos de todas as chaves em tempo $O(n)$ construindo uma tabela T com $2n$ células, onde o valor da j-ésima célula $T[j]$ é dado por $\lceil \log(j+1) \rceil + 1$, para $j = 0, \ldots, 2n-1$. Para construir esta tabela em tempo $O(n)$, observamos que o valor da segunda célula é 2, os valores das duas células seguintes são iguais a 3, os valores das quatro células seguintes são iguais a 4 e assim por diante. Por isto, basta executar o seguinte laço:

$T[0] \leftarrow 1;\ i \leftarrow 0;\ j \leftarrow 1;$
Enquanto $j < 2n$ **faça**
    Para $k = j, \ldots, \min\{j + 2^i - 1, 2n-1\}$ **faça**
        $T[j] \leftarrow i;$
    Fim-Para;
    $j \leftarrow j + 2^i;\ i \leftarrow i + 1;$
Fim-Enquanto

Para calcular o posto de a_i, utilizamos a seguinte proposição.

Proposição 3.20 *Dada uma chave a_i de $\mathcal{A}$, se $c(a_i) < 2^{\lceil \log n \rceil - t}$, então o posto de a_i é dado por*

$$r(a_i) = T[\lfloor c(a_i) \times 2^t \rfloor].$$

Em caso contrário, se $2^{\lceil \log n \rceil - t} \leq c(a_i) < 1$, então o posto de a_i é dado por

$$r(a_i) = T[\lfloor c(a_i) \times 2^{t-\lceil \log n \rceil} \rfloor] + \lceil \log n \rceil.$$

Prova. Primeiro, consideramos o caso onde $c(a_i) < 1/2^t$. Neste caso, temos

$$0 \leq c(a_i) \times 2^t < 1 \;\Rightarrow\; \lfloor c(a_i) \times 2^t \rfloor = 0.$$

Além disso, pela definição de posto, temos $r(a_i) = 1$. Como $T[0] = \lceil \log(0+1) \rceil + 1 = 1$, esta proposição vale para este caso.

Agora, suponha que $1/2^t \leq c(a_i) < 2^{\lceil \log n \rceil - t}$. Seja r tal que $1/2^{t-r+2} \leq c(a_i) < 1/2^{t-r+1}$. Neste caso, pela definição de posto, temos $r(a_i) = r \leq \lceil \log n \rceil + 1$. Além disso, temos

$$\begin{aligned}
2^{r-2} &\leq c(a_i) \times 2^t < 2^{r-1} \\
r-2 &\leq \log(c(a_i) \times 2^t) < r-1 \\
r-2 &= \lfloor \log(c(a_i) \times 2^t) \rfloor \\
&= \lfloor \log \lfloor (c(a_i) \times 2^t) \rfloor \rfloor \\
&= \lceil \log(\lfloor (c(a_i) \times 2^t) \rfloor + 1) \rceil - 1 \\
r &= \lceil \log(\lfloor (c(a_i) \times 2^t) \rfloor + 1) \rceil + 1 = T[\lfloor (c(a_i) \times 2^t) \rfloor].
\end{aligned}$$

No caso em que $r > \lceil \log n \rceil + 1$, a demonstração é quase idêntica. A única diferença é que multiplicamos as duas primeiras desigualdades da seqüência anterior por $2^{-\lceil \log n \rceil}$. Com isto, obtemos que

$$r - \lceil \log n \rceil = T[\lfloor c(a_i) \times 2^{t-\lceil \log n \rceil} \rfloor]$$

no final desta seqüência. □

Observe que o tempo necessário para calcular o posto de cada chave de $\mathcal{A}$ utilizando T é $O(1)$. Na subseção 3.6.4, mostramos como calcular o posto de a_i em tempo $O(1)$ para $\alpha = 2^{1/p}$, onde p é um número inteiro maior de que 1. Isto é necessário porque o algoritmo ECPB obtém fatores de aproximação menores para valores de α próximos a 1. Neste caso, sempre podemos obter uma solução $(2+\epsilon'+o(1))$-aproximada para qualquer $\epsilon' > 0$ utilizando esta implementação o PBPC. Para isto, aplicamos o Corolário 3.19 para o maior valor de $\epsilon \leq \epsilon'$ tal que $p = 1/\log(1+\epsilon/4)$ é inteiro. Isto sempre é possível porque $1/\log(1+\epsilon/4) \to \infty$ quando $\epsilon \to 0$ por valores

positivos.

3.6.2 Análise da Função Construir

Antes de apresentar a nossa implementação para a Função Construir, analisamos uma implementação imediata a partir do pseudocódigo dado pela Tabela 3.4. Se o subvetor L recebido como argumento pela Função Construir tem $|L| \leq K$, então uma ABB ótima é construída através de um dos algoritmos por programação dinâmica descritos na subseção 3.4. Neste caso, a Função gasta um tempo $O(|L|^3)$. Por outro lado, se o subvetor L tem $|L| > K$, então a Função Construir percorre o vetor L para obter as chaves cujo posto é $r(L)$, gastando um tempo $O(|L|)$. Seja $k(L)$ o número de chaves no subvetor L cujo posto é $r(L)$. Todas as outras operações executadas pela Função Construir, incluindo as chamadas à Função Caso-Médio ou Pior-Caso e excluindo as chamadas recursivas, gastam um tempo $O(k(L))$. Por isto, a complexidade de total das chamadas à Função Construir é dada por

$$O\left(\sum_{L\in Z_K} k(L) + \sum_{L\in Z_K} |L| + \sum_{L\in(Z-Z_K)} |L|^3\right), \tag{3-49}$$

onde Z e Z_K são definidos no início da seção 3.5.4.

Como os subvetores de $(Z - Z_K)$ não se sobrepõem, temos

$$\sum_{L\in(Z-Z_K)} |L|^3 = O(nK^2),$$

que é linear em n quando K é fixo. Além disso, como cada chave tem posto $r(L)$ em no máximo um subvetor $L \in Z_K$, temos

$$\sum_{L\in Z_K} k(L) = O(n).$$

Daí, concluímos que todas as chamadas à Função construir gastam um tempo total $O(n)$ exceto pela parcela $\sum_{L\in Z_K} |L|$ de (3-49), que corresponde às buscas pelas chaves de posto mínimo. Nesta implementação, esta busca pode gastar um tempo $\Theta(n \log n)$, em algumas instâncias. Para reduzir a complexidade desta busca, evitamos percorrer todas as chaves do subvetor L em busca das $O(k(L))$ chaves de posto $r(L)$. Realizando esta tarefa em tempo $O(k(L))$, obtemos uma complexidade total $O(n)$. Para isto, implementamos mais um pré-processamento.

3.6.3 Pré-processamento

Primeiro, construímos $t+1$ listas duplamente encadeadas $R_1, \ldots, R_{t+1}$. Na lista R_i, inserimos todas as chaves do vetor de entrada $\mathcal{A}$ cujo posto é i ordenados pelas respectivas posições em $\mathcal{A}$. Claramente, todos as listas podem ser construídas em uma única passada sobre o vetor $\mathcal{A}$, gastando um tempo $O(n)$.

Em seguida, aplicamos o pré-processamento proposto em [30] para o *Problema de Consultas de Mínimo em Intervalos* (*Range Minimum Query* ou RMQ) para o vetor de postos $[r(a_1), r(a_2), \ldots, r(a_n)]$. Dado um vetor de números $[x_1, x_2, \ldots, x_n]$ e índices $i < j$, o RMQ consiste em encontrar um elemento de valor mínimo no subvetor $[x_i, x_{i+1}, \ldots, x_j]$. Em [30], Bender e Farach-Colton mostraram que qualquer consulta pode ser respondida em tempo $O(1)$ depois de aplicado um pré-processamento sobre o vetor $[x_1, x_2, \ldots, x_n]$. Este pré-processamento gasta um tempo $O(n)$. Logo, para qualquer subvetor L, todas as chaves de L cujo posto é $r(L)$ podem ser obtidas em tempo $O(k(L))$ da seguinte forma.

Passo 1: Encontrar uma chave de L cujo posto é mínimo em tempo $O(1)$, utilizando o algoritmo de Bender;

Passo 2: percorrer as $k(L)$ chaves de L cujo posto é $r(L)$, utilizando a lista duplamente encadeada $R_{r(L)}$.

3.6.4 Cálculo de Postos em Escalas Fracionárias

Agora, seja p um número inteiro positivo maior do que 1. Mostraremos como calcular os postos de todas as chaves de $\mathcal{A}$, para $\alpha = 2^{1/p}$ e $t = p\lceil \log n^2 \rceil + p - 1$, gastando um tempo $O(n)$. Para isto, utilizaremos uma tabela U com p valores constantes. Nesta tabela, temos $U[l] = \alpha^l$, para $l = 0, 1, \ldots, p-1$. Como p é fixo e estes valores não dependem da instância, assumimos que eles já estão calculados no início da execução do algoritmo.

Seja $r'(a_i)$ o posto da chave a_i quando o fator de escala é 2 (ao invés de α) e o número total de postos é $t' = \lceil \log n^2 \rceil$. Chamamos o $r'(a_i)$ de *posto auxiliar* da chave a_i. Observe que os postos auxiliares de todas as chaves de $\mathcal{A}$ podem ser calculados em tempo $O(n)$, como descrito na subseção 3.6.1. Agora, considere a seguinte proposição.

Proposição 3.21 *Para toda chave a_i do vetor de entrada $\mathcal{A}$, temos $r'(a_i) = \left\lceil \frac{r(a_i)}{p} \right\rceil$.*

Prova. Dividimos em três casos:

(i) $r(a_i) = 1$;

(ii) $2 \leq r(a_i) \leq p-1$;

(iii) $r(a_i) \geq p$.

Demonstramos apenas o caso (iii), pois os demais casos são quase idênticos. Seja a_i uma chave de $\mathcal{A}$ cujo posto é $r(a_i)$. Pelas definições de t e t', temos $t/p = t' + 1 - 1/p$. Logo, pela definição de posto, obtemos que

$$\begin{aligned}
1/\alpha^{t-r(a_i)+2} &\leq c(a_i) < 1/\alpha^{t-r(a_i)+1} \\
1/2^{t/p-r(a_i)/p+2/p} &\leq c(a_i) < 1/2^{t/p-r(a_i)/p+1/p} \\
-t/p + r(a_i)/p - 2/p &\leq \log c(a_i) < -t/p + r(a_i)/p - 1/p \\
-t' + r(a_i)/p + (1-1/p) - 2 &\leq \log c(a_i) < -t' + r(a_i)/p - 1 \\
-t' + \lceil r(a_i)/p \rceil - 2 &\leq \log c(a_i) < -t' + \lceil r(a_i)/p \rceil - 1 \\
1/2^{t'-\lceil r(a_i)/p \rceil+2} &\leq c(a_i) < 1/2^{t'-\lceil r(a_i)/p \rceil+1}
\end{aligned}$$

Daí, concluímos que o posto auxiliar de a_i é $r'(a_i) = \left\lceil \frac{r(a_i)}{p} \right\rceil$. Para os demais casos, a única diferença é que os limites inferiores dos custos são substituídos por zero desde o início (caso (i)) ou apenas na última linha (caso (ii)). □

Pela proposição anterior, uma vez que o valor do posto auxiliar $r'(a_i)$ de uma chave a_i é calculado, a_i passa a ter apenas p postos possíveis, dados por $p\,r'(a_i) - p + 1, p\,r'(a_i) - p + 2, \ldots, p\,r'(a_i)$. Neste caso, para calcular $r(a_i)$ de forma mais eficiente, construímos mais uma tabela U' durante o pré-processamento. Para calcular $U'[j] = 1/2^{t'-j+1}$, para $j = 1, \ldots, t'$, gastamos um tempo $O(\log n)$. Observe que o intervalo de custos definido para as chaves de posto $r(a_i) = p\,r'(a_i) - l$ tem como limite superior estrito o valor

$$\frac{1}{\alpha^{t-p\,r'(a_i)+l+1}} = \frac{1}{2^{t/p+1/p-r'(a_i)+l/p}} = \frac{1}{2^{t'+1-r'(a_i)+l/p}} = \frac{U'[r'(a_i)]}{U[l]},$$

para $l = 0, 1, \ldots, p-1$. Por isto, se l é o maior índice entre 0 e $p-1$ tal que $c(a_i) < U'[r'(a_i)]/U[l]$, então concluímos que $r(a_i) = p\,r'(a_i) - l$. Como só precisamos testar um número fixo de valores para l, o posto de a_i pode ser calculado em tempo $O(1)$.