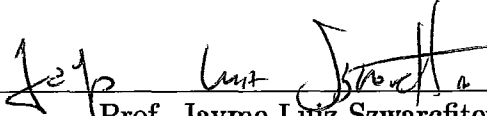


ÁRVORES PARES: UM ESQUEMA PARA DETECÇÃO DE ERROS EM  
ÁRVORES TIPO HUFFMAN

Paulo Eustáquio Duarte Pinto

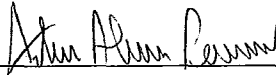
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO  
DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA  
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS  
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE  
DOUTOR EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

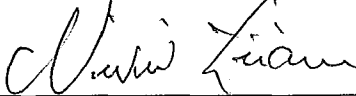
Aprovada por:

  
Prof. Jayme Luiz Szwarcfiter, Ph.D.

  
Prof. Fábio Protti, D.Sc.

  
Prof. Alexandre Sztajnberg, D.Sc.

  
Prof. Artur Alves Pessoa, D.Sc.

  
Prof. Nivio Ziviani, Ph.D.

  
Prof. Valmir Carneiro Barbosa, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 2006

PINTO, PAULO EUSTÁQUIO DUARTE

Árvores pares: Um esquema para detecção de erros em árvores tipo Huffman [Rio de Janeiro] 2006

X, 118 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2006)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Compactação de dados

I. COPPE/UFRJ II. Título ( série )

Para *Nêda, Pedro e Luciana*

## AGRADECIMENTOS

*À minha família, pelo incentivo e apoio constantes.*

*Aos Prof. Jayme L. Szwarcfiter e Prof. Fábio Protti, pela orientação competente e amiga. E pelas lições da generosidade com que conduzem sua vida profissional.*

*Aos colegas da UERJ que acompanharam de perto minha trajetória no doutorado.*

*Ao Prof. Artur Alves Pessoa, pela contribuição dada à tese e pela participação na banca.*

*Ao Prof. Alexandre Sztajnberg, pelas colaborações na UERJ e pela participação na banca.*

*Aos Prof. Nívio Ziviani e Prof. Valmir C. Barbosa, pela participação na banca.*

*Aos Professores e colegas da linha de Algoritmos e Combinatória da COPPE, pelo excelente ambiente acadêmico, do qual pude desfrutar todos esses anos.*

*E aos meus amigos do Escravos da Mauá, fonte inesgotável de alegria e energia para a vida.*

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

## ÁRVORES PARES: UM ESQUEMA PARA DETECÇÃO DE ERROS EM ÁRVORES TIPO HUFFMAN

Paulo Eustáquio Duarte Pinto

Maio/2006

Orientadores: Jayme Luiz Szwarcfiter

Fábio Protti

Programa: Engenharia de Sistemas e Computação

Árvores pares correspondem a códigos de prefixos baseados em Huffman, onde todas as codificações têm paridade par. Esses códigos têm a propriedade de detectar a ocorrência de um número ímpar de erros de 1-bit em uma mensagem codificada. Sua definição foi motivada por um problema proposto por Hamming, em 1980. Neste trabalho caracterizamos árvores pares ótimas considerando duas situações: probabilidades uniformes e probabilidades arbitrárias. Árvores pares ótimas podem ser construídas por um algoritmo  $O(n^3)$  em tempo, ao passo que códigos quase-ótimos podem ser construídos por uma heurística  $O(n \log n)$ , cuja árvore resultante tem custo aproximadamente 5% maior que a correspondente árvore de Huffman, para grande número de símbolos. É desenvolvido um modelo probabilístico para avaliar a capacidade de detecção de erros de árvores pares e são apresentados resultados experimentais com a compressão de vários conjuntos de dados, incluindo linguagem natural. Finalmente, árvores pares são estendidas para árvores  $\alpha$ -pares, que podem ter um aumento expressivo na capacidade de detecção de erros, baseado em um parâmetro  $\alpha$ . É também dado um algoritmo para obter árvores  $\alpha$ -pares ótimas e resultados experimentais com essas árvores.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

EVEN TREES: A SCHEMA TO DETECT ERRORS IN HUFFMAN-TYPE  
TREES

Paulo Eustáquio Duarte Pinto

May/2006

Advisors: Jayme Luiz Szwarcfiter

Fábio Protti

Department: Systems Engineering and Computer Science

Even trees correspond to Huffman-based prefix codes where all encodings have even parity. They have the property of being able to detect the occurrence of an odd number of 1-bit errors introduced in a coded message. Its definition was motivated by a problem posed by Hamming in 1980. In this work we characterize optimal even trees considering two situations: uniform probabilities and arbitrary probabilities. Optimal even trees can be constructed by an  $O(n^3)$  time algorithm, whereas nearly optimal even codes can be constructed by an  $O(n \log n)$  heuristics whose resulting tree has a cost in practice 5% greater than the cost of the corresponding Huffman tree, for a large number of symbols. It is developed a probabilistic model to evaluate the error detection capability of even trees. The work includes experimental results with the compression of different sets of data, including natural language. Finally, even trees are extended to  $\alpha$ -even trees, that can have an increased capacity of error detection, based on a parameter  $\alpha$ . It is also given an algorithm to obtain optimal  $\alpha$ -even trees and various experimental results with these trees.

# Sumário

<b>Lista de Figuras</b>	<b>x</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Huffman e Hamming</b>	<b>7</b>
2.1 Conceituações básicas . . . . .	7
2.2 O código de Huffman . . . . .	11
2.3 O código de Hamming . . . . .	15
2.3.1 Detecção de Erros . . . . .	16
2.3.2 Correção de Erros . . . . .	18
<b>3 Códigos e Árvores Pares</b>	<b>20</b>
3.1 Árvores Hamming-Huffman . . . . .	20
3.2 Códigos e árvores pares . . . . .	22
<b>4 Árvores pares ótimas</b>	<b>27</b>
4.1 Propriedades básicas preliminares . . . . .	27
4.2 Árvores pares ótimas para probabilidades uniformes . . . . .	30
4.2.1 Árvores pares equilibradas e árvores pares ótimas . . . . .	30
4.2.2 Caracterização de árvores pares ótimas . . . . .	38
4.2.3 Comparação de árvores ótimas com árvores de Huffman . . . . .	46
4.3 Árvores pares ótimas para probabilidades arbitrárias . . . . .	50
<b>5 Heurísticas para árvores pares</b>	<b>53</b>
5.1 Heurística 1 . . . . .	53

5.2	Heurística 2 . . . . .	54
5.3	Limites de custo . . . . .	58
5.4	Resultados experimentais . . . . .	65
<b>6</b>	<b>Detecção de Erros em Árvores Pares</b>	<b>72</b>
6.1	Um modelo probabilístico para detecção de erros em árvores pares	72
6.2	Resultados experimentais na detecção de erros . . . . .	77
6.3	Compressão de linguagem natural usando árvores pares . . . . .	78
<b>7</b>	<b>Árvores <math>\alpha</math>-pares</b>	<b>83</b>
7.1	Árvores 1-pares cheias, subcheias e $k$ -completas . . . . .	85
7.2	Árvores 1-pares ótimas para probabilidades uniformes . . . . .	88
7.3	Comparação entre árvores de Huffman e árvores $\alpha$ -pares ótimas para probabilidades uniformes . . . . .	98
7.4	Árvores $\alpha$ -pares ótimas para probabilidades arbitrárias . . . . .	100
7.5	Resultados Experimentais . . . . .	104
<b>8</b>	<b>Conclusões</b>	<b>110</b>



# Lista de Figuras

2.1	Árvore de representação completa de um código para 3 símbolos	11
2.2	Árvores de Huffman para 5 símbolos . . . . .	13
3.1	Árvore Hamming-Huffman para 5 caracteres . . . . .	21
3.2	Árvores pares para 1, 2 e 3 símbolos . . . . .	22
4.1	Árvore par equilibrada para 11 símbolos . . . . .	31
4.2	Árvores pares ótimas para 2, 3 e 4 símbolos . . . . .	42
4.3	Árvore par ótima e árvore de Huffman para 8 símbolos . . . . .	52
5.1	Árvore de Huffman e árvore par da Heurística 1 . . . . .	54
5.2	Árvore de Huffman antes e após Melhoria I . . . . .	55
5.3	Árvore de Huffman antes e após a Melhoria II . . . . .	56
5.4	Árvore par após cada etapa da Melhoria III . . . . .	57
5.5	Árvores de Huffman para $n = 5$ . . . . .	59
5.6	Árvore de Huffman para $n > 5$ , $d_n = 1$ . . . . .	61
5.7	Árvore de Huffman para $n > 5$ , $d_n = 2$ . . . . .	61
5.8	Família especial de árvores de Huffman . . . . .	63
7.1	Exemplos de árvores 0-pares, 0.5-pares e 1-pares . . . . .	85
7.2	Árvore 1-par cheia de profundidade 3 . . . . .	86
7.3	Árvore 1-par subcheia de profundidade 3, com $n = 3$ . . . . .	86
7.4	Árvore 1-par 2-completa de profundidade 5, com $n = 5$ . . . . .	87
7.5	Ampliação do nível de profundidade 2 de árvore 1-par . . . . .	90
7.6	Eliminação da folha de codificação $s_1$ na árvore 1-par . . . . .	91
7.7	Eliminação da folha de codificação $s_4$ na árvore 1-par . . . . .	91



# Capítulo 1

## Introdução

A compactação de dados é uma técnica que visa criar uma representação alternativa econômica de dados, em relação a uma representação padrão.

Trata-se de uma tarefa que nasceu antes dos computadores, pois já no século 19 tivemos a invenção do código Morse, por Samuel Morse. Para transmitir letras pelo telégrafo, elas eram codificadas de forma binária, usando traços e pontos. Morse atribuiu sequências curtas para letras que ocorrem com mais frequência no inglês, como o  $e(.)$  ou  $i(..)$ , e sequências mais longas para letras pouco frequentes, como o  $x(-.-)$  ou  $q(- - .-)$ .

Também no século 19 foi inventado o código Braille, destinado a deficientes visuais. Este código usou a frequência das palavras para prover compactação. Um texto codificado é uma sequência de matrizes  $2 \times 3$  de pontos, onde cada ponto pode estar em relevo ou não, o que permite 64 combinações diferentes para cada matriz. Vinte e seis combinações referem-se a letras e, as demais, a palavras frequentes ou marcações especiais.

Uma idéia básica para a compactação é representar dados com codificações de tamanho variável, tal que o tamanho de uma codificação seja inversamente relacionado à frequência de ocorrência do símbolo correspondente. Essa idéia norteou os primeiros métodos de compactação voltados para a computação, os métodos de Shanon-Fano [9] e de Huffman [15].

Nos primórdios, os métodos apoiados em estatísticas consideravam apenas

a frequência dos símbolos como base para a compactação. Entretanto, não é apenas este aspecto dos dados que permite a compactação. Outras propriedades estruturais podem ser exploradas para esse fim. Por exemplo, a variação dos símbolos na sequência pode ser utilizada. Se tivermos a seguinte sequência de números: 1 2 3 4 5 e representarmos as diferenças entre números consecutivos, a partir do segundo, teríamos a nova sequência 1 1 1 1. Então a compactação poderia ser baseada no primeiro símbolo e na sequência de diferenças. Essa idéia é efetivamente utilizada no padrão JPEG para compressão de imagens. Mas muitas outras propriedades estruturais podem também ser consideradas.

O enfoque de compactação de dados tomou duas direções, conforme haja ou não a necessidade de reprodução fiel dos dados originais, após a descompactação. Há situações em que é necessário reproduzir exatamente os dados originais, como é o caso da compactação de textos e de programas. Nesta hipótese, o método de compactação é *sem perdas*. Por outro lado, em aplicações multimídia, envolvendo imagens e sons, pode não haver problemas em pequenas perdas após a descompactação. Então, pode-se adotar a compactação *com perdas*.

Vejam os um breve histórico do desenvolvimento dos métodos de compactação sem perdas, segundo Witten et al [47]. O *método de Huffman* [15], criado na década de 1950, predominou na computação durante algumas décadas. A compactação obtida era a de representar um caractere com 5 bits, em média, ao invés dos 8 da codificação ASCII. Uma grande revolução ocorreu nos fins da década de 1980. Primeiro com as sucessivas propostas de Ziv-Lempel [50, 51], que levou aos métodos de dicionário *LZ77* e *LZ78*, adaptativos. Mais tarde essas propostas foram aperfeiçoadas por Welch [46], chegando-se ao *método LZW*, que teve grande aceitação. Paralelamente, foi aperfeiçoada por Pasco e Rissanen [29, 38] a *Codificação Aritmética* que tornou-se uma técnica complementar a vários métodos.

Na Codificação Aritmética, a compactação de uma sequência de símbolos resulta em um único número no intervalo  $[0,1]$ , através de um processo de

refinamento sucessivo.

Os novos enfoques adaptativos utilizam um dicionário tal que as repetições de cadeias do texto vão sendo tabeladas nesse dicionário, à medida que o texto é lido. Com eles, conseguiu-se melhorar a taxa de compressão para menos de 4 bits por caractere. O método LZW tornou-se a base para o padrão GIF de imagens e também para o padrão V.42 da CCITT, para transmissão de dados via Modem.

Na década de 1980 foram ainda criados os métodos de contexto, os melhores métodos existentes, em termos de taxa de compressão. Eles possibilitaram a utilização de pouco mais de 2 bits por caractere. Os métodos dessa linha são o *Prediction by Partial Matching - PPM*, criado por Cleary e Witten [5] e o Burrows Wheeler Transform -BWT, criado por Burrows e Wheeler [4].

O método PPM tem uma certa analogia com o LZW, mudando a forma de busca no dicionário e utilizando codificação aritmética.

Desde a década de 1980 o que se tem feito na área de compactação sem perdas é o aperfeiçoamento dos métodos mencionados. Além disso, o método de Huffman entrou novamente em destaque por apresentar ótimos resultados na compressão de textos, quando se considera como base para a compressão as palavras do texto e não seus caracteres [53, 47].

Excelentes descrições dos métodos de compactação sem perdas, com bastante detalhes, podem ser encontradas em [40, 41, 47]. Estudos comparativos desses métodos foram apresentados em [36, 47].

Há dois contextos importantes onde a compactação de dados associa-se a outras operações sobre os dados: na transmissão de dados e na recuperação de informações.

Na transmissão de dados a vantagem da compactação relaciona-se ao menor tempo de transmissão que ela possibilita. Como o meio físico por onde trafegam os dados compactados sempre está sujeito a ruídos, torna-se necessário adotar um controle de erros.

Segundo Tanenbaum [44], os projetistas de redes adotam duas estratégias

distintas para controlar erros: *correção de erros* e *detecção de erros*. Na correção de erros são incluídas informações redundantes em cada bloco enviado, tal que os dados iniciais possam ser reconstruídos, em caso de erro. Na detecção de erros, o número de informações redundantes é menor e, quando um erro é detectado, é feita uma solicitação ao transmissor para a retransmissão dos dados errados.

Cada uma dessas técnicas combina-se com diferentes tecnologias de transmissão: em canais muito confiáveis, como fibra ótica, pode-se adotar a detecção de erros; em canais pouco confiáveis, como é o caso atual da comunicação sem fio, deve-se adotar a correção de erros.

Na recuperação de informações, especialmente no contexto da Internet, a compactação de dados é um imperativo econômico, pois os volumes de dados são astronômicos. Considere, por exemplo, os grandes sites de busca. Segundo Baeza e Ribeiro [2], alguns requisitos adicionais são também fundamentais nessas situações.

O primeiro deles é que é necessário o acesso randômico aos dados compactados, para que buscas específicas possam ser feitas de forma eficiente. A maioria dos bons métodos de compressão precisa decodificar os dados desde o início de um bloco, para se ter acesso a todos os dados. É o caso da Codificação Aritmética, que é inviável para esses fins. Considerando ainda a necessidade de buscas eficientes, é mais adequado trabalhar-se com palavras ao invés de caracteres, na compactação.

As velocidades de compactação e descompactação são também importantes. Mas a velocidade na descompactação é primordial, pois dados são compactados uma única vez e buscados milhares de vezes.

Outra característica importantíssima do método de compactação neste contexto é a possibilidade de que algumas buscas possam ser feitas diretamente no texto compactado, compactando algum parâmetro dessa busca. Como consequência, é possível buscar mais rapidamente em textos compactados, pois há muito menos texto a ser processado.

Outra consideração importante é a memória computacional exigida.

Baeza e Ribeiro [2] apresentam uma comparação dos métodos de compactação, considerando os requisitos comentados. Essa comparação está mostrada na Tabela 1.1, que é autoexplicativa.

	Cod. Aritmética	Huffman (caractere)	Huffman (palavra)	LZW
Taxa de compactação	Muito alta	Baixa	Muito alta	Alta
Velocidade compactação	Baixa	Alta	Alta	Muito alta
Velocidade descompactação	Baixa	Alta	Muito alta	MuitoAlta
Quantidade de Memória	Baixa	Baixa	Alta	Média
Busca compactada	Não	Sim	Sim	Sim
Busca randômica	Não	Sim	Sim	Não

Tabela 1.1: Comparação de métodos de compactação na recuperação de informações

Considerando a tabela mostrada, verifica-se a importância do método de Huffman, quando aplicado a palavras, no contexto de Recuperação de Informações.

Nesta tese propomos uma variante do método de Huffman, as *árvores pares*, que objetivam integrar a compactação com o controle de erros, na descompactação.

Esse método seria uma alternativa para as tarefas similares que ocorrem nos processos de transmissão de dados e que são executadas de forma independente. Os dados codificados com este novo método já contêm redundâncias tal que a ocorrência de erros introduzidos durante a transmissão pode ser detectada no momento da descompressão. Poderia também ser utilizado para aumentar a confiabilidade no armazenamento dados em meios de baixa confiabilidade.

Os principais resultados obtidos são os seguintes:

a) a caracterização teórica das árvores pares quando se considera probabilidades uniformes e probabilidades arbitrárias.

b) um algoritmo polinomial para a construção de árvores pares ótimas, com complexidade de tempo  $O(n^3)$  e de espaço,  $O(n^2)$ .

c) uma heurística para a construção de árvores pares quase ótimas, com o correspondente algoritmo polinomial, com complexidade de tempo  $O(n \log n)$  e de espaço,  $O(n)$ .

d) um modelo probabilístico para a detecção de erros em árvores pares.

e) a extensão da estrutura de árvores pares para árvores  $\alpha$ -pares, onde o nível de detecção de erros pode ser ajustado. É também apresentado um algoritmo polinomial para a construção de árvores  $\alpha$ -pares, com complexidade de tempo  $O(n^3)$  e de espaço,  $O(n^2)$ .

A organização desta tese é a seguinte: no Capítulo 2 fazemos uma descrição dos métodos de controle de erros, com ênfase no código de Hamming, e também uma descrição detalhada do método de compactação de Huffman e suas variantes. No Capítulo 3 apresentamos o problema que motivou o presente trabalho, proposto por Hamming em 1980, referente à integração das tarefas de controles de erros e compactação. Apresentamos sua solução delineada, as árvores Hamming-Huffman e a nova alternativa, as árvores pares. No Capítulo 4 discorremos sobre a construção de árvores pares ótimas. No Capítulo 5, uma heurística para a construção de árvores pares quase ótimas, apresentando uma excelente taxa de aproximação. No Capítulo 6 desenvolvemos um modelo probabilístico para a avaliação do poder de detecção de erros de árvores pares. No Capítulo 7 apresentamos uma extensão das árvores pares, as árvores  $\alpha$ -pares, que permitem o ajuste do nível de detecção de erros desejado. Finalmente, o Capítulo 8 é destinado a conclusões e perspectivas futuras.

Partes desta tese foram objeto de várias publicações. Versões preliminares e simplificadas dos Capítulos 3 e 4 foram publicadas em [31, 32, 33]. Uma versão anterior da seção de Árvores Pares ótimas para o caso de probabilidades uniformes do Capítulo 4 foi publicada em [35]. Uma versão anterior do Capítulo 4 e parte do Capítulo 5 foram publicados em [34]. Finalmente, um resumo dos Capítulos 4 a 6 foi submetido para publicação [37].



# Capítulo 2

## Huffman e Hamming

Neste capítulo apresentaremos resumos do método de Huffman para compactação de dados e dos métodos de controle de erros em dados codificados, com ênfase no código de Hamming. Iniciaremos estabelecendo as conceituações básicas utilizadas no restante da tese.

### 2.1 Conceituações básicas

Uma *Árvore enraizada*  $T$ , ou simplesmente *árvore*, é um conjunto finito de elementos denominados nós ou vértices tais que

1.  $T = \emptyset$ , e a árvore é dita vazia, ou
2. Existe um nó especial,  $r$ , chamado *raiz de  $T$* ; os restantes constituem um único conjunto vazio ou são divididos em  $m \geq 1$  conjuntos disjuntos não vazios, as *subárvores de  $r$* , ou simplesmente *subárvores*, cada qual, por sua vez, uma árvore.

Dentre as diversas representações possíveis para árvores, utilizaremos neste trabalho apenas a representação hierárquica.

Vejamos uma série de definições de parentesco em árvores:

Seja  $v$  o nó raiz da subárvore  $T_v$  de  $T$ . Os nós raízes  $w_1, w_2, \dots, w_j$  das subárvores de  $T_v$  são chamados *filhos de  $v$* ;  $v$  é chamado *pai de  $w_1, w_2, \dots, w_j$* . Os nós  $w_1, w_2, \dots, w_j$  são *irmãos*. Se  $z$  é *filho de  $w_1$* , então  $w_2$  é *tio de  $z$*  e  $v$  é

*avô* de  $z$ . O número de filhos de um nó é denominado o *grau de saída do nó*; Se  $x$  pertence à subárvore  $T_v$ ,  $x$  é denominado *descendente de  $v$*  e  $v$ , *ancestral de  $x$* . Neste caso, sendo  $x \neq v$ ,  $x$  é *descendente próprio de  $v$*  e  $v$ , *ancestral próprio de  $x$* . Um nó que não possui descendentes próprios é chamado *folha*. Um nó não folha é dito *interior* ou *interno*.

Vejamos uma série de conceituações relativas a caminhos em árvores:

Uma sequência de nós distintos  $v_1, v_2, \dots, v_k$ , tal que existe sempre, entre nós consecutivos ( $v_1$  e  $v_2, v_2$  e  $v_3, \dots, v_{k-1}$  e  $v_k$ ) a relação ‘*é filho de*’ ou ‘*é pai de*’, é denominada um *caminho da árvore*. Diz-se que  $v_1$  *alcança  $v_k$*  e vice-versa. O valor  $(k - 1)$  é o *comprimento do caminho*. Se  $v_1$  é a raiz então  $v_1, v_2, \dots, v_k$  é um *caminho raiz* e se  $v_k$  é uma folha, então  $v_1, v_2, \dots, v_k$  é um *caminho raiz-folha*. Denomina-se *comprimento de caminho*,  $P_T(n)$  de uma árvore  $T$  com  $n$  folhas, a soma de todos os caminhos raiz-folha de  $T$ . Quando não houver ambiguidade escreveremos simplesmente  $P(n)$ . *Nível de um nó  $v$*  é o número de nós do caminho da raiz até o nó  $v$ . O nível da raiz é, portanto, 1. A *altura de um nó  $v$*  é o número de nós do maior caminho de  $v$  até um de seus descendentes. As folhas, portanto, têm altura 1. A *altura de uma árvore  $T$*  é igual ao nível máximo de seus nós. Representa-se a altura de  $T$  por  $h(T)$ , enquanto  $h(v)$  é a altura da subárvore de raiz  $v$ . A *profundidade de um nó  $v$*  é o comprimento do caminho da raiz até esse nó. A raiz, portanto, tem profundidade 0. A *profundidade de uma árvore  $T$*  é igual à profundidade máxima de seus nós. Representa-se a profundidade de uma árvore  $T$  por  $d(T)$ , e a profundidade do nó  $v$  por  $d(v)$ .

Deve ser notado que o conceito de altura de árvore é levemente diferente, nas referências utilizadas, [16, 6]. Para o primeiro autor, a altura de uma folha é considerada 1, enquanto para o segundo, 0. Estamos, neste particular, adotando a nomenclatura de [16]. Ainda em termos de nomenclatura, usaremos indistintamente  $h(T)$  ou  $h$  para nos referirmos à altura de  $T$  e  $d(T)$  ou  $d$  para nos referirmos à profundidade de  $T$ .

Vejamos, a seguir, conceituações relativas a árvores binárias:

Uma *árvore binária*  $T$  é um conjunto finito de elementos denominados nós ou vértices, tal que

1.  $T = \emptyset$ , e a árvore é dita vazia, ou

2. Existe um nó especial  $r$ , denominado *raiz de  $T$* , e os restantes podem ser divididos em dois subconjuntos disjuntos,  $T_r^E$  e  $T_r^D$ , a *subárvore esquerda de  $r$*  e a *subárvore direita de  $r$* , respectivamente, as quais são também árvores binárias.

Vejamos uma conceituação especial de parentesco nas árvores binárias:

A raiz da subárvore esquerda (direita) de um nó  $v$ , se existir, é denominada *filho esquerdo (direito) de  $v$* . Qualquer um dos filhos pode existir, sem a existência do outro.

Utilizaremos, também, conceituações especiais para alguns tipos de árvores binárias:

Uma *árvore estritamente binária* é uma árvore binária em que cada nó possui 0 ou 2 filhos. Uma *árvore binária completa* é aquela que apresenta a seguinte propriedade: se  $v$  é um nó tal que alguma subárvore de  $v$  é vazia, então  $v$  se localiza ou no último ou no penúltimo nível da árvore. Uma *árvore binária cheia* é aquela em que, se  $v$  é um nó com alguma de suas subárvores vazias, então  $v$  se localiza no último nível da árvore.

Finalmente, vejamos alguns conceitos relativos a códigos.

Seja  $\mathcal{S}$  um conjunto de  $n$  elementos, chamados *símbolos*. A cada símbolo  $s_i \in \mathcal{S}$  associa-se uma probabilidade de ocorrência  $f_i$ .

Ao longo deste trabalho, assumimos  $f_i \leq f_{i+1}$ , para  $1 \leq i < n$ . Geralmente consideramos que a soma das probabilidades  $f_1, \dots, f_n$  seja 1, embora este fato não seja necessário para a otimalidade dos códigos.

Uma *codificação  $e$*  para um símbolo  $s \in \mathcal{S}$  é uma sequência finita de 0's e 1's, associada a  $s$ . Cada 0 e 1 é um *bit* de  $e$ . Os bits da codificação  $e$  são rotulados  $1, 2, \dots, l$ , e  $l$  é o *tamanho* de  $e$ . Um *segmento  $e(i, j)$*  é uma subsequência de  $e$ , começando no bit  $i$  e terminando no bit  $j$ . Um segmento

$e(1, j)$  é um *prefixo* de  $e$ . A *paridade* de  $e$  é a paridade da quantidade de 1's contida em  $e$ . O conjunto de codificações para todos os símbolos de  $\mathcal{S}$  é um *código*  $\mathcal{C}$  para  $\mathcal{S}$ . Dois códigos  $\mathcal{C}, \mathcal{C}'$  para  $\mathcal{S}$  são *equivalentes* quando existir uma bijeção  $\mathcal{C} \Leftrightarrow \mathcal{C}'$ , tal que as codificações correspondentes tenham mesmo tamanho. O *custo* de um código,  $c$  é dado pela soma  $c = \sum_{i=1}^n t_i f_i$ , onde  $t_i$  é o tamanho da codificação relativa ao símbolo  $s_i$  e  $f_i$ , sua frequência. Um código é *ótimo* quando seu custo é mínimo. Um código no qual cada codificação não coincida com um prefixo de nenhuma outra codificação é um *código de prefixo*.

Uma *mensagem*  $M$  é uma sequência de símbolos. A *mensagem codificada* de  $M$  é a sequência correspondente de codificações. A *paridade* de uma mensagem codificada é o número de 1's contido nas codificações.

Vejamos algumas definições que relacionam códigos com árvores binárias.

Dada uma árvore binária  $T$ , as arestas que levam a filhos esquerdos são rotuladas com 0, enquanto aquelas que levam a filhos direitos, são rotuladas com 1. A *paridade* de um nó  $z$  é a paridade da quantidade de 1's nas arestas do caminho da raiz até  $z$ . Um nó é *par* ou *ímpar*, de acordo com sua paridade, respectivamente.

Uma *árvore binária de representação* de um código  $\mathcal{C}$  é uma árvore binária  $T$ , tal que exista uma correspondência um a um entre as codificações  $e \in \mathcal{C}$  e os caminhos raiz-folha  $p$  de  $T$ , de tal forma que  $e$  é precisamente a sequência de rótulos, 0 ou 1, das arestas que formam  $p$ . Um código admite uma árvore binária de representação se e somente se é um código de prefixo. Uma *árvore de representação completa* de  $\mathcal{C}$  é uma árvore binária  $T^*$  obtida da árvore de representação  $T$  de  $\mathcal{C}$ , pela adição de uma nova folha como o segundo filho de cada nó que possua exatamente um filho. As folhas originais de  $T$  são *folhas de codificação*, ao passo que as folhas recentemente introduzidas são *folhas de erro*.

Dado um código  $\mathcal{C}$ , seu custo é exatamente o custo de uma árvore binária de representação completa,  $T$ , definido como  $c(T) = \sum_{i=1}^n f_i d_i$ , onde  $d_i$  é

a profundidade da folha de codificação correspondente ao símbolo  $s_i$  e  $f_i$  a probabilidade desse símbolo. Uma árvore de representação ótima é aquela de custo mínimo.

Símbolo	Frequência	Codificação
a	0.5	0
b	0.4	11
c	0.1	100

Tabela 2.1: Código de prefixo para 3 símbolos

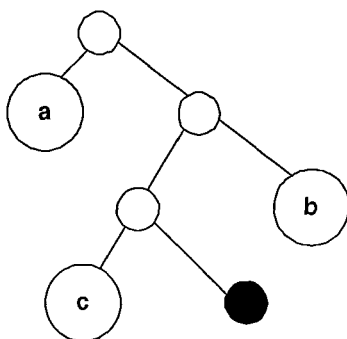


Figura 2.1: Árvore de representação completa de um código para 3 símbolos

A Tabela 2.1 ilustra um código  $\mathcal{C}$  de prefixo contendo 3 símbolos, e a figura 2.1, uma árvore binária,  $T$ , de representação completa para  $\mathcal{C}$ . O custo de  $T$  é dado por  $c(T) = 1 * 0.5 + 2 * 0.4 + 3 * 0.1 = 1.6$ . Na figura mostrada foi acrescentada uma folha especial (folha de erro), como irmã da folha correspondente ao símbolo  $c$ .

## 2.2 O código de Huffman

O código de Huffman [15] constitui um dos mais tradicionais métodos de compactação. Esse código tem dois aspectos importantes: é um código de prefixos e trabalha com codificações de tamanho variável.

Como é um código de prefixos, dado um código de Huffman (ótimo), então

existe uma árvore binária respectiva, a *árvore de Huffman*. Nessa árvore, para cada símbolo existe uma folha correspondente.

O procedimento de Huffman para gerar uma árvore ótima é baseado em dois princípios:

- (1) Em uma árvore de Huffman ótima, dados dois símbolos  $s_i, s_j$ , com probabilidades  $f_i, f_j$  e profundidades  $d_i, d_j$  respectivas, então  $f_i \geq f_j$  implica  $d_i \leq d_j$ , ou seja, símbolos com maior probabilidade estão mais próximos da raiz.
- (2) Em uma árvore de Huffman, os dois símbolos de menor frequência têm mesma profundidade e sempre podem ser tornados irmãos.

Dado um conjunto de símbolos  $\{s_1, s_2, \dots, s_n\}$ , com probabilidades respectivas  $\{f_1, f_2, \dots, f_n\}$ , o procedimento para gerar uma árvore de Huffman consiste em criar uma floresta inicial contendo  $n$  subárvores triviais (contêm apenas o nó raiz), respectivamente uma para cada símbolo, e executar  $n - 1$  passos de agregação. Em cada passo de agregação, as subárvores de menor peso são unidas através de uma nova raiz comum. O peso de uma subárvore é a soma das probabilidades de suas folhas. Esse algoritmo tem complexidade de tempo  $O(n \log n)$  e, de espaço,  $O(n)$ .

Por exemplo, dado o conjunto de 5 símbolos  $\{s_1, s_2, s_3, s_4, s_5\}$ , com probabilidades respectivas  $\{0.4, 0.2, 0.2, 0.1, 0.1\}$ , as duas árvores da Figura 2.2 são árvores de Huffman. A árvore da esquerda foi construída da seguinte forma: inicialmente foram agregados os símbolos  $s_4$  e  $s_5$ . Em seguida, agregaram-se  $s_2$  e  $s_3$ . No terceiro passo, foi feita a fusão das duas primeiras agregações e, no passo final, esse resultado foi unido a  $s_1$ . Para construir a árvore da direita, houve uma mudança no segundo passo, agregando-se o resultado da junção de  $s_4$  com  $s_5$  a  $s_3$ , o que era também possível pelo algoritmo.

O método de Huffman obtém um código que minimiza a soma  $\sum_{i=1}^n f_i d_i$ , o comprimento médio de uma codificação. Este método é bastante eficiente e, embora muito simples, foi objeto de um grande número de extensões e

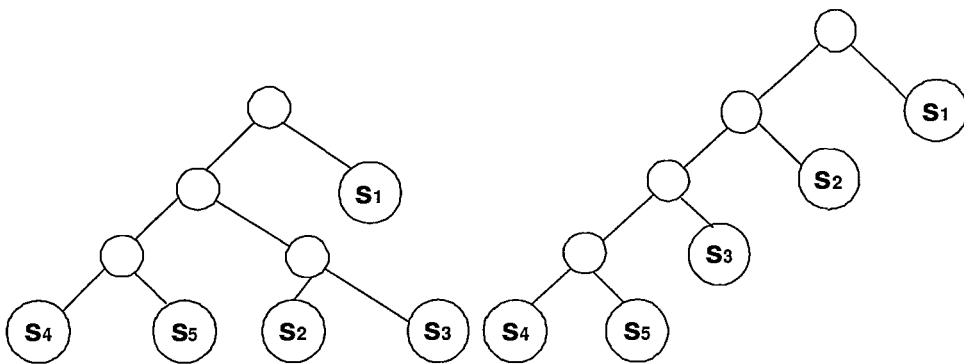


Figura 2.2: Árvores de Huffman para 5 símbolos

variações ao longo dos anos, algumas das quais serão comentadas a seguir. Boas revisões da literatura sobre essas extensões e variantes podem ser encontradas em [1, 21].

Knuth [17] estudou com profundidade as árvores de Huffman. Alguns autores caracterizaram essas árvores para situações especiais. Por exemplo, Gutman [12] caracterizou a árvore para certos tamanhos de alfabeto, onde a distribuição é tal que  $f_i$  é proporcional a  $1/i$ .

A árvore de Huffman não é única, para dada situação, conforme visto no exemplo da Figura 2.2. Schwartz [42] descreveu a árvore de altura mínima e Buro [3], a de altura máxima.

Alguns autores consideraram problemas ligados a restrições lexicográficas no código gerado. Um dos problemas tratados é o de gerar a árvore de Huffman tal que as folhas da árvore se relacionem ordenadamente, da esquerda para a direita, com o conjunto dado de símbolos. Os primeiros a considerarem esse problema foram Gilbert e Moore [11]. Outra importante variante são os códigos canônicos de Huffman, utilizados quando se tem um alfabeto muito grande e se necessita decodificação muito rápida, como é o caso dos sites de busca da Internet, onde ao invés da compactação ser baseada em caracteres, ela é baseada em palavras, obtendo-se resultados muito bons [47]. Esses códigos dispensam a árvore de decodificação e são obtidos a partir dos tamanhos das codificações na árvore de Huffman. Um dos primeiros trabalhos nessa linha

foi o de Schwartz e Kallick [43]. Moffat e Katajainen [45] desenvolveram um algoritmo de decodificação, baseado na representação canônica, cujo comportamento é linear em tempo e espaço.

Outra modificação na codificação Huffman, muito importante no contexto de recuperação de informações, foi apresentada por Moura, Navarro, Ziviani e Baeza-Yates [27]. O código gerado utiliza bytes ao invés de bits, a forma tradicional. O que se perde com o aumento das codificações é compensado de forma muito vantajosa pelo ganho de flexibilidade e velocidade na busca de dados compactados. Uma descrição detalhada do método e seus algoritmos é apresentada em [53]. Resultados relacionados são apresentados em [28, 52].

Uma importante extensão foi a da construção adaptativa de árvores de Huffman. Nesta extensão, para se compactar um conjunto de dados, ele não é examinado a priori, para se determinar as probabilidades dos símbolos. Estas vão sendo descobertas dinamicamente. O método inicial foi apresentado por Faller [8] e, mais tarde, de forma independente, por Galaggar [10]. Posteriormente Knuth [19] provou a correção do método e efetuou melhorias no algoritmo, que passou a se chamar FGK. Milidiú, Laber e Pessoa [22] melhoraram a análise do algoritmo.

O algoritmo para o método de Huffman adaptativo é baseado na importante propriedade ‘sibling’. Essa propriedade afirma que uma árvore de representação de um código de prefixo é uma árvore de Huffman se e somente se ela puder ser reorganizada tal que, um percurso em nível na árvore resultante obtém os nós ordenados em forma não crescente de pesos. As únicas transformações permitidas, nesse caso, são trocas entre si de subárvores esquerdas com as correspondentes subárvores direitas, em qualquer ponto da árvore. Essas transformações não alteram o custo da árvore, evidentemente. Por exemplo, na Figura 2.2, a árvore da direita tem essa propriedade, pois a sequência de pesos no percurso em nível é 1 , 0.6 , 0.4 , 0.4 , 0.2 , 0.2 , 0.2 , 0.1 , 0.1 . Já a árvore da esquerda não tem a propriedade. Entretanto, se trocarmos entre si as subárvores do nível 3, obtemos uma árvore equivalente,



com a propriedade mencionada.

A construção paralela de códigos de Huffman foi tratada por Milidiú, Laber e Pessoa [23].

Uma questão prática importante pode ser a restrição do tamanho máximo das codificações, para considerar as questões de hardware, especialmente o tamanho da palavra, que atualmente é de 32 bits. A construção de árvores de Huffman com restrição de tamanho foi considerada por Turpin e Moffat [26], Larmore e Hirschberg [20] e Milidiú e Laber [24, 25].

Segundo Lelewer e Hirschberg [21], tem sido dada pouca atenção ao problema de detecção e correção de erros em conexão aos códigos de Huffman. O principal enfoque ligado a essa questão é o de códigos de sincronização. A idéia é procurar códigos de Huffman tal que, quando ocorre um erro, ou seja, a troca de um bit em uma mensagem codificada, esse erro não se propague muito. Um dos trabalhos nessa linha é o de Rudner [39].

Considerando a observação acima, o trabalho desta tese situa-se em uma área pouco explorada e pode contribuir para a discussão do problema.

Na próxima seção faremos um resumo dos métodos de detecção e correção de erros em códigos.

## 2.3 O código de Hamming

O controle de erros é necessário na transmissão ou no armazenamento de dados em certos tipos de periféricos, dada a insuficiente confiabilidade envolvida. Normalmente esse controle é feito anexando redundâncias aos dados codificados, tal que a presença de erros possa ser detectada. Dependendo do enfoque utilizado, faz-se a nova obtenção dos dados ou correção dos erros, isto é, pode-se adotar apenas um enfoque de detecção, que é mais simples, ou de correção, que exige maior redundância.

De certa forma, o controle de erros atua de forma inversa à compactação de dados.

Esta seção trata resumidamente dos métodos de tratamento de erros e é fortemente baseada em [13, 40, 44]. Uma característica importante, descoberta empiricamente sobre os tipos de erros introduzidos em transmissões de dados, é que eles não se distribuem aleatoriamente ao longo de uma mensagem, mas costumam ocorrer em ‘rajadas’, isto é, afetam partes consecutivas da mesma.

Nas estratégias de controle de erro, o conceito de *Distância Hamming* entre duas codificações desempenha um papel importante.

Dadas duas codificações  $c_1$  e  $c_2$ , de mesmo comprimento  $n$ , o número de bits nos quais essas codificações diferem entre si é chamado *distância Hamming* e, se a distância é  $d$ , o significado é o de que são necessárias  $d$  trocas de bits em  $c_1$  para se obter  $c_2$ .

Cada estratégia de tratamento de erros é brevemente descrita a seguir.

### 2.3.1 Detecção de Erros

Para que um código tenha a capacidade de detectar  $d$  erros, é necessário que a distância Hamming entre qualquer par de codificações seja, pelo menos,  $d + 1$ . Por exemplo, para se detectar apenas 1 erro, não podemos ter um par onde as codificações difiram entre si em apenas 1 bit, pois a troca desse bit pode transformar uma codificação na outra e o erro não ser detectado.

A forma mais primitiva de detecção de erros é por eleição. Tratando-se de transmissão de dados, uma mesma mensagem seria enviada um certo número (ímpar) de vezes e sempre seria adotada como correta aquela que correspondesse ao maior número de coincidências.

Duas outras formas têm sido, entretanto, mais utilizadas: O uso de bits de paridade e bits de verificação.

Bits de paridade horizontal são agregados a cada  $m$  bits de uma mensagem codificada, e referem-se à paridade do número de bits 1 contidos nos  $m$  bits anteriores. Bits de paridade vertical são  $m$  bits anexados ao final de  $k$  conjuntos de  $m$  bits, dando, cada um deles, a paridade do número de bits de mesma posição 1 a  $m$  dos  $k$  conjuntos. Algumas vezes se usa paridade dupla, tanto

horizontal quanto vertical.

O uso de bits de cheque consiste em anexar  $k$  bits a cada conjunto de  $m$  bits, tal que muitas das configurações de  $m + k$  bits sejam proibidas. Quando uma configuração proibida é detectada, há a indicação do erro.

Na prática, o método usual e que se tornou padrão é o de *Código Polinomial*, também conhecido como *Código de Redundância Cíclica (CRC)*. Este método consiste em se acrescentar a uma mensagem um número  $k$  de bits de redundância, obtidos da forma descrita a seguir. A mensagem inicial, com  $n$  bits, é associada a um polinômio  $P(x)$  de grau  $n - 1$ , sendo o bit mais à esquerda associado ao termo  $x^{n-1}$  e o mais à direita, a  $x^0$ . Utiliza-se um polinômio padrão  $G(x)$ , de grau  $k$ ,  $k < n - 1$  e os bits de redundância são associados ao polinômio  $R(x)$ , que representa o resto da divisão, módulo 2, de  $P(x)$  por  $G(x)$ .  $R(x)$  tem, evidentemente, grau  $k - 1$ . Por exemplo, se tivermos uma mensagem codificada como '1100101' e, como polinômio,  $G(x) = x^3 + x^1$ , correspondendo a '101', obtemos como resto o polinômio  $R(x) = x^1 + 1$ , correspondente a '011'. A mensagem a ser transmitida, então passa a ser '1100101011'. Na recepção, utiliza-se o mesmo  $G(x)$  para testar se o valor recebido apresenta o mesmo resto informado.

O poder de detecção do método descrito varia com o valor de  $k$  e com o polinômio escolhido como referência. Se  $G(x)$  contém dois ou mais termos, todos os erros de um bit são detectados. Para a detecção de dois erros, é necessário que  $G(x)$  não divida polinômios da forma  $x^p + 1$ , onde  $p$  é a diferença de posições dos erros. Se  $G(x)$  divide  $x + 1$ , todos os erros em número ímpar de bits são detectados. Finalmente, o mais importante, todos os erros tipo rajada de até  $k$  bits são detectados.

Certos polinômios tornaram-se padrões internacionais. O que é usado no padrão IEEE 802 é:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

Ele captura todos os erros tipo rajada de tamanho até 32, e todas as rajadas que afetam um número ímpar de bits.

### 2.3.2 Correção de Erros

Na estratégia de correção de erros, o objetivo é, ao final da transmissão, detectar a ocorrência de erros e sua localização exata, o que permite a correção de dados errados. Aqui são usados os Códigos de Correção de Erros, dos quais os *Hamming Codes* são os mais conhecidos.

Em um código de correção de erros, para se detectar e corrigir  $d$  erros é necessário que as codificações tenham distância mínima  $2d+1$  pois elas estarão tão distantes entre si que, mesmo com  $d$  mudanças, a codificação original é a que estará menos distante da alteração, podendo ser unicamente determinada.

O esquema de Hamming Codes detecta e corrige erros de 1 bit e consiste em se agregar bits de paridade para combinações de posições da codificação original tal que, quando a recepção detecta erro, consegue-se obter a posição errada pela soma das posições dos bits de paridade que indicaram erro.

A Tabela 2.2 mostra a recodificação da palavra ‘TESTE’, originalmente em ASCII (7 bits), e passando para 11 bits, sendo os “Hamming bits” aqueles das posições 1, 2, 4 e 8, na recodificação (numeração da esquerda para a direita). O bit 1 checa a paridade dos bits 3, 5, 7, 9, 11; o bit 2, de 3, 6, 7, 10, 11; o bit 4, de 5, 6, 7 e o bit 8, de 9, 10, 11.

Caractere	ASCII	Cod. 11 bits
	.....	12..4.....8.....
T	1010100	00110101100
E	1000101	10100000101
S	1010011	01110100011
T	1010100	00110101100
E	1000101	10100000101

Tabela 2.2: Recodificação de ‘TESTE’ de ASCII para Hamming Code.

Embora a técnica de Hamming Codes só sirva para corrigir erros de 1 bit, ela pode ser adaptada para correção de erros do tipo rajada, os mais comuns em transmissão. Isso é feito mudando-se a forma de transmissão. Os dados são colocados em uma matriz de dimensão  $k * n$  e a transmissão é feita coluna a

coluna. Na recepção a matriz é reconstruída e, se houver ocorrido uma rajada de até  $k$  bits, cada linha pode ser individualmente recuperada.

Outros tipos de códigos de correção fazem uso intensivo de álgebra computacional, tais como os códigos de Golay, Reed-Muller e Reed-Salomon. Descrições dos mesmos podem ser encontradas em [7, 14, 48].

No próximo capítulo discutiremos duas propostas de integração entre a compactação de dados e a detecção de erros.

# Capítulo 3

## Códigos e Árvores Pares

Neste capítulo apresentamos o problema que motivou esta tese, o modelo proposto por Hamming e a proposta alternativa feita por nós dos códigos e árvores pares.

### 3.1 Árvores Hamming-Huffman

Como foi visto nas duas seções anteriores, compactação de dados e detecção ou correção de erros são tarefas realizadas em momentos diferentes e usando técnicas distintas, nos processos de transmissão de dados. Uma pergunta natural é aquela de ser possível ou não integrar esses dois aspectos do tratamento de dados.

Essa pergunta foi apresentada por Hamming no livro *Coding and Information Theory* [13]. Ele propõe como modelo, mas sem se estender muito, a estrutura de dados mostrada a seguir, que ele denominou, Árvores Hamming-Huffman (AHH, de agora em diante).

Estas estruturas permitiriam o desenvolvimento de métodos para integrar as duas tarefas mencionadas, tendo-se os benefícios da compactação de Huffman com a proteção contra ruídos de Hamming. A idéia é que a própria codificação de dados contivesse redundâncias que permitissem a detecção de determinados tipos de erro. Na sua proposta, as AHH levariam a uma codificação tal, que todos os erros de 1 bit nas codificações pudessem ser detectados

na recepção. Basicamente, a idéia é a de proibir certas codificações que, quando presentes na recepção, indicariam o erro.

De forma análoga às Árvores de Huffman, as AHH seriam árvores estritamente binárias, contendo as codificações em folhas, mas haveria a introdução de folhas de erro tal que, para cada codificação, todas as trocas de 1 bit levem a folhas de erro, no processo de decodificação.

Veamos a idéia através do exemplo apresentado no livro citado, onde supõe-se probabilidades uniformes. A Tabela 3.1 mostra os símbolos com suas codificações respectivas e, a Figura 3.1, a árvore Hamming-Huffman correspondente.

Símbolos	Codificação
a	000
b	0110
c	1010
d	1100
e	1111

Tabela 3.1: Exemplo de um código Hamming-Huffman

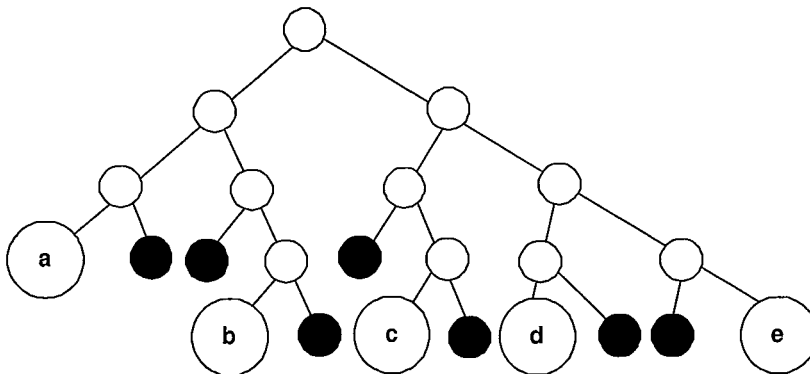


Figura 3.1: Árvore Hamming-Huffman para 5 caracteres

No exemplo mostrado, os prefixos ‘001’, ‘010’, ‘0111’, ‘100’, ‘1011’, ‘1101’ e ‘1110’ são prefixos proibidos e, quando ocorrerem na decodificação, indicam troca de 1 bit em dada codificação.

Hamming deixa o problema de desenvolver essa idéia como uma tarefa para o leitor. Até onde pesquisamos, não foram encontradas referências na literatura sobre a questão, muito importante, pelo menos do ponto de vista teórico.

Esta tese pretende abordar este problema. Embora a ausência de referências tenha trazido a oportunidade de um tratamento original ao tema, também exigiu um grande esforço para o desenvolvimento de todo o esquema de análise necessário.

Na próxima seção apresentamos os códigos e árvores pares, nossa proposta de solução do problema apresentado por Hamming.

## 3.2 Códigos e árvores pares

Nesta seção, descrevemos os códigos pares, que integram as funções de compactação e detecção de erros. Esses códigos têm a propriedade de detectar qualquer número ímpar de erros em uma mensagem codificada, mas também podem detectar significativamente erros pares.

Um *código de paridade* é um código de prefixo no qual todas as codificações de mesmo tamanho têm mesma paridade. Um *código par* é um código no qual todas as codificações têm paridade par. Analogamente, uma *árvore par* é uma árvore de representação completa para um código par. A Figura 3.2 ilustra exemplos de árvores pares.

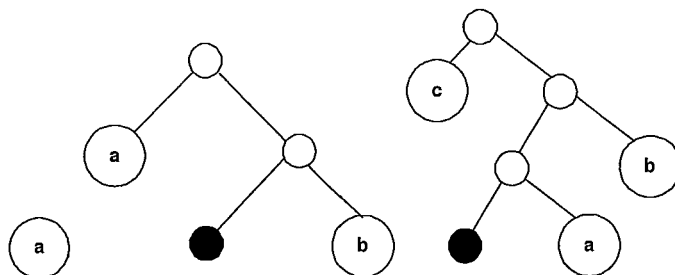


Figura 3.2: Árvores pares para 1, 2 e 3 símbolos

A proposição a seguir relaciona códigos de paridade com códigos pares.



**Teorema 3.2.1** *Seja  $\mathcal{C}$  um código de paridade para um conjunto de símbolos  $\mathcal{S}$ . Existe um código par  $\mathcal{C}'$  para  $\mathcal{S}$ , tal que  $\mathcal{C}, \mathcal{C}'$  são equivalentes.*

**Demonstração:** Seja  $\mathcal{C}$  um código de paridade para  $\mathcal{S}$ . Construa  $\mathcal{C}'$  a partir de  $\mathcal{C}$ , da forma seguinte. Ordene as codificações de  $\mathcal{C}$  por tamanho e faça o seguinte procedimento: Sejam  $d_1, d_2, \dots, d_k$  os tamanhos distintos de codificações aparecendo em  $\mathcal{C}$ . Para  $i = 1, \dots, k$ , tome as codificações de tamanho  $d_i$  e, caso aquelas codificações tenham paridade ímpar, troque o  $i$ -ésimo bit em todas as codificações com tamanho igual ou maior que  $d_i$ . Seja  $\mathcal{C}'$  o código obtido após a iteração  $k$  deste procedimento. É claro que todas as codificações de  $\mathcal{C}'$  são pares. Além disso,  $\mathcal{C}$  sendo um código de prefixos implica que  $\mathcal{C}'$  também deve ser. Para verificarmos esta afirmativa, sejam  $e', e''$  duas codificações de  $\mathcal{C}$ ,  $|e'| \leq |e''|$ . Então, sempre que o bit  $j$  de  $e'$  muda, no procedimento acima, o bit  $j$  de  $e''$  também muda. Consequentemente,  $e' \neq e''(1, |e'|)$ , implicando que as codificações correspondentes de  $\mathcal{C}'$  são também distintas. Isto é,  $\mathcal{C}'$  é de fato um código de prefixos e consequentemente um código par. ■

A proposição anterior implica que, para o nosso propósito, podemos restringir nossa atenção a códigos pares, embora muitas das propriedades tenham validade para a classe mais ampla de códigos de paridade.

Podemos concluir que códigos pares detectam a ocorrência de um número ímpar de erros em uma mensagem, da seguinte forma. Sabemos que todas as codificações são pares, então uma mensagem codificada também é par. A introdução de um número ímpar de bits errados, a mensagem codificada torna-se ímpar. Como as codificações são pares, isso implica que uma folha de erro vai ser atingida na árvore de representação completa do código ou então o processo de decodificação termina em algum nó de paridade ímpar da árvore. A detecção de erros pares será tratada no Capítulo 6.

Um código no qual todas as codificações são ímpares é um *código ímpar* e sua árvore de representação, uma *árvore ímpar*. Para um código  $\mathcal{C}$ , o simétrico

de  $\mathcal{C}$  é um código obtido de  $\mathcal{C}$  pela troca do primeiro bit de todas as codificações. Claramente, um código é ímpar se e somente se seu simétrico é par.

Deve-se notar que códigos ímpares não têm a propriedade relatada antes. Por exemplo, se tivermos um código ímpar  $\mathcal{C} = \{1, 01\}$  e uma mensagem 01, se o primeiro bit for mudado para 1, transformando a mensagem codificada para 11, o processo de decodificação decodificará essa mensagem sem sinalizar erros.

Podemos fazer algumas comparações entre os códigos Hamming-Huffman e os códigos pares. Códigos pares são uma ampliação da idéia proposta por Hamming, já que, como um código par detecta todos os erros em número ímpar, em particular detecta erros de um bit, propriedade básica dos códigos Hamming-Huffman. Entretanto, há uma sutil diferença no processo da detecção de erros entre os códigos pares e os códigos Hamming-Huffman. Enquanto que, para os códigos pares, a especificação feita é a de que sejam detectados erros em número ímpar introduzidos na mensagem codificada, sem a preocupação específica com cada codificação, nos códigos Hamming-Huffman a detecção do erro é feita na codificação errada, identificando exatamente onde ocorreu o erro. Embora este fato não esteja explícito nas explicações de Hamming, é perceptível claramente que essa é a intenção da proposta.

Então, mensagens codificadas nesses dois códigos podem ter processos diferentes de detecção de erro de um bit. A tabela 3.2 contém os dois códigos para 5 símbolos, supondo-se probabilidades uniformes: o código  $C_1$ , Hamming-Huffman apresentado na seção anterior e um código par,  $C_2$ . Poderíamos ter os seguintes processos de detecção usando os dois códigos:

a) Suponhamos uma mensagem usando o código HH para 'bae'. Essa mensagem seria codificada como '01100001111'. Se fosse introduzido um erro no segundo bit e ela fosse mudada para '00100001111', o processo de decodificação descobriria, após a análise dos 3 primeiros bits ('001'), que houve um erro, pois nenhuma codificação começa com esse prefixo e que o erro foi no primeiro carácter. Isso ilustra a idéia da detecção ser a nível de caractere.

Símbolo	$C_1(\text{HH})$	$C_2(\text{par})$
a	000	00
b	0110	011
c	1010	1001
d	1100	101
e	1111	11

Tabela 3.2: Comparação código Hamming-Huffman x código par para 5 caracteres

b) Suponhamos a mesma mensagem para o código par. Ela seria codificada como '0110011'. Se também fosse introduzido um erro no segundo bit e ela fosse mudada para '0010011', os dois primeiros bits, '00', seriam decodificados como 'a', os quatro seguintes, '1001', como 'c' e se chegaria ao final sem conseguir decodificar o restante da mensagem, '1'. O erro seria detectado, mas não seria possível dizer qual caractere teve seus bits trocados. Ou seja, não há a possibilidade de detecção a nível de caractere, mas sim da mensagem como um todo.

Embora o código HH, conforme sugerido por Hamming, permita a detecção de erro de um bit na exata codificação onde ele foi introduzido, ele não é um código de correção pois, conforme foi visto anteriormente, para ser um código de correção a distância entre as codificações de mesmo tamanho teria que ser 3, o que não ocorre com as codificações '1010' e '1100', por exemplo.

Desta forma, os códigos pares são uma alternativa vantajosa aos códigos HH pois, além de se proporem a detectar mais tipos de erros, têm menor custo, ou seja, maior taxa de compressão. No exemplo mostrado, o custo do código Hamming-Huffman seria 19/5, enquanto o do código par, 14/5. Isso ocorre porque os códigos pares têm menos configurações de erros.

Um fato interessante é o de que, se relaxarmos a exigência de detecção de erro a nível de caractere, para o código HH, ele se torna exatamente o código par apresentado. Isto pode ser visto notando-se que, nas demonstrações anteriores, as únicas propriedades necessárias eram que o código fosse de prefixo

e que codificações de mesmo tamanho tivessem mesma paridade. Estas propriedades estão presentes no código HH, com a relaxação mencionada.

Quanto ao fato inverso, isto é, códigos pares que detectassem erros a nível de caractere, não é algo possível quando as codificações tenham tamanhos diferentes, pois erros em codificações grandes sempre podem levar a decodificações parcialmente erradas e não há como evitar isso.

No próximo capítulo trataremos de códigos ótimos. Como cada código tem uma árvore de representação correspondente, passaremos a focar os códigos pares através de suas correspondentes árvores pares.

# Capítulo 4

## Árvores pares ótimas

Neste capítulo caracterizamos os códigos e árvores pares ótimos. Inicialmente mostraremos algumas propriedades básicas, deduzidas diretamente da definição de árvores pares.

### 4.1 Propriedades básicas preliminares

Podemos reformular o problema de determinar um código par ótimo, em termos da árvore par de representação correspondente, da seguinte forma:

**Entrada:** Um alfabeto  $\Sigma = \{s_1, s_2, \dots, s_n\}$ ,  $n > 1$ , e um vetor  $(f_1, f_2, \dots, f_n)$  de probabilidades correspondentes.

**Saída:** Uma árvore par,  $T$ , para  $\Sigma$ , que minimiza a função objetivo  $o = \sum_{i=1}^n f_i \cdot d_i$ , onde  $d_i$  é a profundidade da folha de codificação atribuída a  $s_i$  em  $T$ .

Algumas propriedades básicas das árvores pares ótimas podem ser obtidas diretamente a partir da definição. Inicialmente vamos apresentar alguns lemas relativos às folhas de erro.

**Lema 4.1.1** *Numa árvore par ótima,  $T$ , para  $n > 1$ , folhas de erro têm sempre como irmãos, filhos de codificação, à direita.*

**Demonstração:** Quando  $n = 1$  a árvore tem apenas a raiz, não tendo folhas de erro. Suponhamos que, para  $n > 1$ , houvesse em  $T$  uma folha de erro  $l$  que fosse filho direito de um nó interno,  $r$ , que obrigatoriamente tem algum descendente que é folha de codificação. Como  $l$  tem paridade ímpar,  $r$  tem paridade par. Podemos construir  $T'$  a partir de  $T$ , eliminando  $l$  e substituindo  $r$  pela sua subárvore esquerda.  $T'$  seria uma árvore par e teria custo inferior ao de  $T$ , pois as folhas de codificação da subárvore com raiz em  $r$  teriam sua profundidade diminuída de 1. Isso é absurdo.

Logo, as folhas de erro são filhos esquerdos de nós internos, cuja paridade é par.

Suponhamos, agora, que o nó irmão de  $l$  não seja uma folha de codificação e que  $r$  seja o pai de  $l$ . Podemos construir  $T'$  a partir de  $T$  eliminando  $l$ , substituindo  $r$  pela subárvore à sua direita,  $s$ , e trocando as subárvores esquerda e direita de  $s$ .  $T'$  é claramente uma árvore par, com custo inferior a  $T$ , pois as folhas de codificação da subárvore com raiz em  $r$  tiveram sua profundidade diminuída de 1. Isso é absurdo.

Logo, o irmão à direita de  $l$  é uma folha de codificação. ■

**Corolário 4.1.2** *Em um código par ótimo, para  $n > 1$ , as codificações de maior comprimento sempre terminam em 1.*

**Corolário 4.1.3** *Árvores pares ótimas nunca são árvores binárias cheias.*

**Lema 4.1.4** *Numa árvore par ótima,  $T$ , todas as folhas de erro ocorrem nos dois últimos níveis da árvore.*

**Demonstração:** Seja  $T$  uma árvore par ótima, com  $h(T) = q$ . Suponhamos, por absurdo, que haja uma folha de erro  $l$  no nível  $p < q - 1$ . Então poderíamos criar  $T'$  a partir de  $T$  da seguinte forma: inicialmente  $T'$  é uma cópia de  $T$ . Em  $T'$  existe uma subárvore com raiz  $r$  no nível  $q - 1$ , de paridade ímpar e tendo, pelo Lema 4.1.1 como filho esquerdo uma folha de erro e como filho direito, uma folha de codificação. Substituímos em  $T'$ ,  $l$  por  $r$  e  $r$  por uma

folha de erro.  $T'$  tem custo menor que  $T$ , pois a única diferença nas folhas de codificação é que uma delas teve a sua profundidade modificada de  $q$  para  $p + 1 < q$ . Isso é absurdo.

Logo, todas as folhas de erro estão situadas nos dois últimos níveis da árvore par. ■

**Lema 4.1.5** *A árvore par ótima,  $T$ , é única quando o número de símbolos é  $1 \leq n \leq 3$ .*

**Demonstração:** Sejam  $T_1, T_2, T_3$  as árvores pares ótimas para  $n$  de 1 a 3, respectivamente. Essas árvores pares ótimas estão representadas na Figura 3.2. Para  $n = 1$  a afirmação é óbvia e temos  $c(T_1) = 0$ . Para  $n = 2$  e 3 pode-se verificar o resultado por tentativa. O custo dessas árvores ótimas é dado por:

$$c(T_2) = f_1 + 2f_2,$$

$$c(T_3) = f_1 + 2f_2 + 3f_3. \quad \blacksquare$$

Uma *árvore ímpar ótima* para  $n$  símbolos é definida da seguinte forma:

a) Se  $n = 1$  a árvore consiste de uma raiz, tendo duas folhas como filhos.

À esquerda, uma folha de erro  $e$ , à direita, uma folha de codificação.

b) Se  $n > 1$ , ela é obtida a partir de uma árvore par ótima, trocando-se as subárvores esquerda e direita entre si.

Observe que não temos total simetria entre árvores pares ótimas e árvores ímpares ótimas, o que é objeto do próximo lema.

**Lema 4.1.6** *A única árvore par ótima que tem apenas uma codificação na subárvore direita é a árvore par para 2 símbolos.*

**Demonstração:** A árvore par ótima para  $n = 2$  possui apenas uma folha de codificação na subárvore direita. Suponhamos que, em uma árvore par ótima  $T$  para  $n$  símbolos,  $n > 2$ , houvesse apenas uma folha de codificação na subárvore direita e  $n - 1$  na subárvore esquerda. Então a folha de codificação

da direita deveria ter codificação '11', a menor possível. Poderíamos, obter  $T'$  a partir de  $T$ , colocando como subárvore esquerda de  $T'$  apenas uma folha de codificação e como direita, a árvore ímpar ótima obtida a partir da subárvore esquerda de  $T$ .  $T'$  seria uma árvore par com custo inferior a  $T$ , pois apenas uma das profundidades das codificações foi diminuída de 1, em relação a  $T$ . Isso é absurdo.

Logo, a única árvore par ótima que tem apenas uma codificação na subárvore direita é a árvore par ótima para 2 símbolos. ■

A seguir caracterizaremos as árvores pares ótimas para probabilidades uniformes.

## 4.2 Árvores pares ótimas para probabilidades uniformes

Nesta seção estaremos tratando da situação em que as probabilidades dos símbolos são uniformes. Esta condição ficará implícita em todos os casos tratados.

Inicialmente, vamos definir uma classe de árvores pares que será útil adiante.

### 4.2.1 Árvores pares equilibradas e árvores pares ótimas

Uma *árvore par(ímpar) equilibrada* para  $n$  símbolos é uma árvore par(ímpar)  $T$  com as seguintes características:

a) Se  $n \leq 3$  e  $T$  é par, então  $T$  corresponde a uma das árvores da Figura 3.2. Se  $T$  é ímpar, ela só é definida para  $n = 2$  e  $n = 3$  e é a árvore simétrica da árvore par equilibrada correspondente.

b) Se  $n > 3$  então a subárvore esquerda de  $T$  é uma árvore par(ímpar) equilibrada para  $\lfloor n/2 \rfloor$  símbolos e a subárvore direita, uma árvore ímpar(par) equilibrada para  $\lceil n/2 \rceil$  símbolos.



A Figura 4.1 é uma árvore par equilibrada para 11 símbolos.

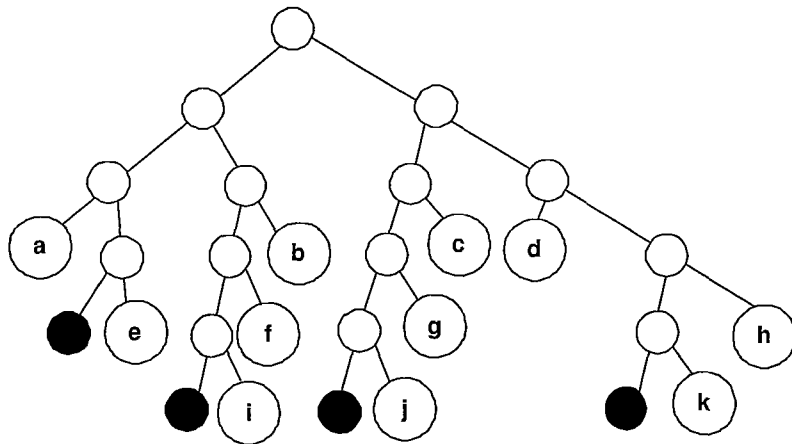


Figura 4.1: Árvore par equilibrada para 11 símbolos

Dada uma árvore par ótima  $T$  para  $n$  símbolos, com probabilidades uniformes,  $f_i = 1/n$ ,  $1 \leq i \leq n$ , temos a relação:

$c(T) = P(T)/n$ , onde  $P(T)$  é o caminho de codificação de  $T$ . Portanto, uma árvore par ótima tem custo mínimo e caminho de codificação mínimo. O caminho de codificação mínimo é dado pelo seguinte teorema:

**Teorema 4.2.1**

$$P(n) = \begin{cases} 0, & \text{se } n = 1 \\ 3, & \text{se } n = 2 \\ n + \min_{1 \leq i \leq n-2} \{P(i) + P(n-i)\}, & \text{se } n > 2 \end{cases}$$

**Demonstração:** Os resultados para  $n = 1$  e  $2$  podem ser verificados a partir da Figura 3.2. Se  $n > 2$ , consideraremos 2 casos:

*Caso 1.* a subárvore esquerda tem somente uma codificação. Então temos:  $P(n) = 1 + (n-1) + P(n-1) = n + P(1) + P(n-1)$ , dado que, a subárvore esquerda tem apenas uma folha de codificação e, a direita é uma árvore ímpar ótima para  $n-1$  símbolos. Observe agora que todas as folhas de codificação dessa subárvore têm profundidade 1 unidade maior, relativamente à raiz.

*Caso 2.* a subárvore esquerda tem mais de uma codificação. Seja  $i$  o número de codificações na subárvore esquerda, que é uma árvore par ótima,

$1 < i \leq n - 2$ . O limite  $n - 2$  é dado pelo Lema 4.1.6. A subárvore direita é uma árvore ímpar ótima com  $n - i$  codificações. Todas as codificações têm profundidade uma unidade a mais, relativamente à raiz. O mínimo valor possível para  $P(n)$  é dado pela minimização  $\min_{1 < i < n-1} \{i + P(i) + (n - i) + P(n - i)\} = n + \min_{1 < i \leq n-2} \{P(i) + P(n - i)\}$ . ■

Essa recorrência permite criar um algoritmo de Programação Dinâmica, já que para o cálculo de um valor ótimo, só é necessário lançar mão do cálculo para valores ótimos menores. Este algoritmo preenche quatro vetores:  $P$ ,  $c$ ,  $k_1$  e  $k_2$ . Em  $P$  é guardado o caminho de codificações; em  $c$  o custo; em  $k_1$  e  $k_2$ , os números mínimo e máximo, respectivamente, de codificações na subárvore esquerda da árvore par ótima.  $k_1$  foi incluído porque em alguns casos temos mais de uma possibilidade, como será mostrado;  $k_2$  indicará o valor mais próximo de  $\lceil n/2 \rceil$ .

### Algoritmo *APOT*( $n$ )

**Início:**

$P[1] \leftarrow 0; \quad c[1] \leftarrow 0; \quad k_2[1] \leftarrow 0; \quad k_1[1] \leftarrow 0;$

**Para**  $i$  **de** 2 **até**  $n$ :

$m \leftarrow P[i - 1]; \quad k_2[i] \leftarrow 1; \quad k_1[i] \leftarrow 1;$

**Para**  $j$  **de** 2 **até**  $\lfloor i/2 \rfloor$ :

**Se**  $((P[j] + P[i - j]) < m)$  **Então**

$m \leftarrow P[j] + P[i - j]; \quad k_2[i] \leftarrow j; \quad k_1[i] \leftarrow j;$

**Senão Se**  $((P[j] + P[i - j]) = m)$  **Então**

$k_2[i] \leftarrow j;$

**Fp;**

$P[i] \leftarrow i + m; \quad c[i] \leftarrow P[i]/i;$

**Fp;**

**Fim;**

O algoritmo só analisa a metade das possibilidades de árvores pares ótimas, deixando de lado as soluções simétricas. Ele tem, evidentemente, complexi-

dade  $O(n^2)$ .

A Tabela 4.1, construída a partir do algoritmo, mostra valores de  $P(n)$ ,  $c(n)$ ,  $k_1(n)$  e  $k_2(n)$ , para  $n$  entre 1 e 60.

n	P	c	$k_1$	$k_2$	n	P	c	$k_1$	$k_2$	n	P	c	$k_1$	$k_2$
1	0	0	0	0	21	102	4.86	9	10	41	239	5.83	17	20
2	3	1.5	1	1	22	108	4.91	10	11	42	246	5.86	18	21
3	2	6	1	1	23	114	4.96	11	11	43	253	5.88	19	21
4	10	2.5	1	2	24	120	5	12	12	44	260	5.91	20	22
5	14	2.7	2	2	25	127	5.08	12	12	45	267	5.93	21	22
6	18	3	3	3	26	134	5.15	12	13	46	274	5.96	22	23
7	23	3.29	3	3	27	141	5.22	12	13	47	281	5.98	23	23
8	28	3.5	3	4	28	148	5.29	12	14	48	288	6	24	24
9	33	3.67	3	4	29	155	5.34	12	14	49	296	6.04	24	24
10	38	3.8	4	5	30	162	5.4	12	15	50	304	6.08	24	25
11	43	3.91	5	5	31	169	5.45	12	15	51	312	6.12	24	25
12	48	4	6	6	32	176	5.5	12	16	52	320	6.15	24	26
13	54	4.15	6	6	33	183	5.55	12	16	53	328	6.19	24	26
14	60	4.29	6	7	34	190	5.59	12	17	54	336	6.22	24	27
15	66	4.4	6	7	35	197	5.63	12	17	55	344	6.25	24	27
16	72	4.5	6	8	36	204	5.67	12	18	56	352	6.29	24	28
17	78	4.59	6	8	37	211	5.70	13	18	57	360	6.32	24	28
18	84	4.67	6	9	38	218	5.74	14	19	58	368	6.34	24	29
19	90	4.7	7	9	39	225	5.77	15	19	59	376	6.37	24	29
20	96	4.74	8	10	40	232	5.8	16	20	60	384	6.4	24	30

Tabela 4.1: Dados de árvores pares ótimas para probabilidades uniformes

Podemos observar alguns pontos em relação a essa tabela:

a) Os vetores  $k_1$  e  $k_2$  permitem construir, de forma recursiva, a árvore par, levando em conta que as subárvores ótimas de uma árvore ótima são também ótimas.

b) De uma maneira geral, há mais de uma árvore par ótima para dado  $n$ . O vetor  $k_1$  mostra isso, parcialmente. Em particular, para  $n = 4$ , há duas

árvores ótimas possíveis e este é o maior valor de  $n$  para o qual a árvore ótima tem apenas um codificação na subárvore esquerda.

c) Para  $n > 4$ , sempre que  $k_1, k_2 < \lceil \frac{n}{2} \rceil$  existe também uma árvore equilibrada com esse valor de codificações na subárvore da direita.

d) Para números da forma  $n = 3 \cdot 2^k + q$ ,  $q \in \{-1, 0, 1\}$ ,  $k \geq 1$ , inteiro, parece haver apenas um tipo de árvore ótima.

e) Pelo que se observa, a árvore par equilibrada parece ser sempre uma árvore par ótima, no caso de probabilidades uniformes. Esta idéia é confirmada a seguir.

Primeiramente, calculemos o caminho de codificação,  $P(n)$ , de uma árvore equilibrada par,  $T$ .

**Teorema 4.2.2** *Seja  $P(n)$  o caminho de codificação de uma árvore equilibrada par,  $T$ , para  $n$  símbolos. Então:*

$$P(n) = \begin{cases} 0, & \text{se } n = 1 \\ 3, & \text{se } n = 2 \\ n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}, & \text{se } n > 2 \end{cases}$$

**Demonstração:** Se  $n \leq 5$ , podemos verificar, por inspeção, que o teorema é correto. Para  $n > 5$ , usamos indução em  $n$ . Assumamos que a expressão acima esteja correta para  $2 < j < n$ . Aplicando a definição e a hipótese de indução, temos:

$$P(n) = n + P(\lfloor n/2 \rfloor) + P(\lceil n/2 \rceil)$$

onde

$$P(\lfloor n/2 \rfloor) = \lfloor n/2 \rfloor \left( \left\lceil \log \frac{\lfloor n/2 \rfloor}{3} \right\rceil + 3 \right) - 3 \cdot 2^{\lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil}$$

e

$$P(\lceil n/2 \rceil) = \lceil n/2 \rceil \left( \left\lceil \log \frac{\lceil n/2 \rceil}{3} \right\rceil + 3 \right) - 3 \cdot 2^{\lceil \log \frac{\lceil n/2 \rceil}{3} \rceil}.$$

Temos, então, 2 casos:

*Caso 1:*  $n$  é da forma  $n = 3 \cdot 2^{q-1} + 1$ . Então:

$$\lceil \log \frac{n}{3} \rceil = q, \lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil = q - 1, \lceil \log \frac{\lceil n/2 \rceil}{3} \rceil = q - 2, \lfloor n/2 \rfloor = 3 \cdot 2^{q-2}.$$

Consequentemente:

$$P(n) = n + \lfloor n/2 \rfloor (q - 2 + 3) - 3 \cdot 2^{q-2} + \lceil n/2 \rceil (q - 1 + 3) - 3 \cdot 2^{q-1}$$

Ou seja,

$$P(n) = n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}.$$

*Caso 2:*  $n$  é da forma  $n = 3 \cdot 2^q$  ou  $n = 3 \cdot 2^{q-1} + p$ ,  $1 < p < 3 \cdot 2^{q-1}$ . Então:

$$\lceil \log \frac{n}{3} \rceil = q, \lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil = \lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil = q - 1.$$

Consequentemente:

$$P(n) = n + \lfloor n/2 \rfloor (q - 1 + 3) - 3 \cdot 2^{q-1} + \lceil n/2 \rceil (q - 1 + 3) - 3 \cdot 2^{q-1}$$

Ou seja,

$$P(n) = n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}.$$

Nos dois casos, o resultado também é válido para  $n$ . ■

Finalmente, temos:

**Teorema 4.2.3** *Árvores pares equilibradas são ótimas.*

**Demonstração:** Seja  $T$  uma árvore par equilibrada para  $n$  símbolos. Para  $n \leq 6$ , podemos verificar, por inspeção, que o teorema é correto. Para  $n > 6$  usaremos indução. Pelo Teorema 4.2.2,  $P(n) = n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}$ . A hipótese de indução assegura que árvores pares equilibradas para menos de  $n$  símbolos são ótimas. Para o passo de indução, seja  $T'$  uma árvore ótima para  $n > 6$  símbolos. Como as subárvores são também ótimas, pela hipótese de indução podemos substituir as subárvores  $T'_L$  e  $T'_R$  de  $T'$  por árvores pares equilibradas, mantendo  $T'$  ótima. Seja  $i$  o número de símbolos na subárvore esquerda de  $T'$ . Aplicaremos o Teorema 4.2.2 às subárvores.

Vamos provar, inicialmente, que  $i > 2$ . Assumamos  $i = 1$ . Então:

$$P_{T'}(n) = n + (n - 1)(\lceil \log \frac{n-1}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n-1}{3} \rceil}.$$

Consideremos os dois casos seguintes:

*Caso 1:* Se  $\lceil \log \frac{n}{3} \rceil = \lceil \log \frac{n-1}{3} \rceil = q$ , então  $P_{T'}(n) = P_T(n) + (n - q) - 3$ .

Como, neste caso,  $(n - q) > 3$ , temos  $P_{T'}(n) > P_T(n)$ , uma contradição.

*Caso 2:* Se  $\lceil \log \frac{n}{3} \rceil = q$  e  $\lceil \log \frac{n-1}{3} \rceil = q-1$ , então  $P_{T'}(n) = P_T(n) + 3 \cdot 2^{q-1} - (q + 2)$ .

Como, neste caso,  $(q > 1)$  implica  $3 \cdot 2^{q-1} > (q + 2)$ , temos  $P_{T'}(n) > P_T(n)$ , também uma contradição. Consequentemente,  $i > 1$ .

Assumamos agora  $i = 2$ . Então:

$$P_{T'}(n) = n + 3 + (n - 2)(\lceil \log \frac{n-2}{3} \rceil + 3) - 3 \cdot 2^{\lceil \frac{n-2}{3} \rceil}.$$

Examinaremos, novamente, os possíveis casos:

*Caso 1':* Se  $\lceil \log \frac{n}{3} \rceil = \lceil \log \frac{n-2}{3} \rceil = q$ , então  $P_{T'}(n) = P_T(n) + n - (2q + 3)$ .

Claramente, isto só pode ocorrer para  $n > 7$ . Portanto,  $n > (2q + 3)$  e  $P_{T'}(n) > P_T(n)$ , o que é uma contradição.

*Caso 2':* Se  $\lceil \log \frac{n}{3} \rceil = q$  e  $\lceil \log \frac{n-2}{3} \rceil = q-1$ , então  $P_{T'}(n) = P_T(n) + 3 \cdot 2^{q-1} - (2q + 1)$ .

Uma vez que, neste caso,  $(q > 1)$  implica  $3 \cdot 2^{q-1} > 2q + 1$ , temos:  $P_{T'}(n) > P_T(n)$ , novamente uma contradição. Concluimos que  $i \neq 2$ .

Sabemos agora que  $i > 2$ . Reescrevemos  $P_{T'}(n)$  em termos de uma função  $f$  definida no intervalo  $[3, \lfloor \frac{n}{2} \rfloor]$  e vamos provar que  $f(i) \geq f(i + 1)$  para  $i \in [3, \lfloor \frac{n}{2} \rfloor - 1]$ . Consequentemente,  $f(i)$  atinge o mínimo para  $i = \lfloor \frac{n}{2} \rfloor$ , e, então,  $T'$  é uma árvore par ótima. Isto significa que  $T'$  é equilibrada e equivalente a  $T$ . Consequentemente,  $T$  também deve ser ótima. Temos:

$$f(i) = n + i(\lceil \log \frac{i}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{i}{3} \rceil} + (n - i)(\lceil \log \frac{n-i}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n-i}{3} \rceil}.$$

Considerando valores consecutivos para  $i$ ,

$$\begin{aligned}
f(i) - f(i+1) &= n + i(\lceil \log \frac{i}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{i}{3} \rceil} \\
&+ (n-i)(\lceil \log \frac{n-i}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n-i}{3} \rceil} \\
&- (n+(i+1))(\lceil \log \frac{i+1}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{i+1}{3} \rceil} \\
&+ (n-i-1)(\lceil \log \frac{n-i-1}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n-i-1}{3} \rceil}.
\end{aligned}$$

Sejam  $\lceil \log \frac{i}{3} \rceil = q_1$  e  $\lceil \log \frac{n-i}{3} \rceil = q_2$ . Estamos agora diante de quatro casos, porque é possível termos, por um lado,  $\lceil \log \frac{i+1}{3} \rceil = q_1$  ou  $\lceil \log \frac{i+1}{3} \rceil = q_1 + 1$ , e, por outro,  $\lceil \log \frac{n-i-1}{3} \rceil = q_2$  ou  $\lceil \log \frac{n-i-1}{3} \rceil = q_2 - 1$ . Observe que estamos supondo que o número de símbolos na subárvore esquerda nunca é superior ao da direita, ou seja,  $q_2 \geq q_1$ .

*Caso I:* Se  $\lceil \log \frac{i+1}{3} \rceil = q_1$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2$ , então  $f(i) - f(i+1) = q_2 - q_1 \geq 0$ .

*Caso II:* Se  $\lceil \log \frac{i+1}{3} \rceil = q_1$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2 - 1$ , então  $f(i) - f(i+1) = (n-i-1) - 3 \cdot 2^{q_2-1} + q_2 - q_1$ . As igualdades  $\lceil \log \frac{n-i}{3} \rceil = q_2$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2 - 1$  só ocorrem quando  $n-i = 3 \cdot 2^{q_2-1} + 1$ . Então,  $f(i) - f(i+1) = q_2 - q_1 \geq 0$ .

*Caso III:* Se  $\lceil \log \frac{i+1}{3} \rceil = q_1 + 1$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2$ , então  $f(i) - f(i+1) = -i + 3 \cdot 2^{q_1} + q_2 - q_1 - 1$ . As igualdades  $\lceil \log \frac{i}{3} \rceil = q_1$  e  $\lceil \log \frac{i+1}{3} \rceil = q_1 + 1$  só ocorrem quando  $i = 3 \cdot 2^{q_1}$ . Então  $f(i) - f(i+1) = q_2 - q_1 - 1 \geq 0$ , uma vez que, neste caso,  $q_2 > q_1$ .

*Caso IV:* Se  $\lceil \log \frac{i+1}{3} \rceil = q_1 + 1$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2 - 1$ , então  $f(i) - f(i+1) = -2i + n + 3 \cdot 2^{q_1} - 3 \cdot 2^{q_2-1} + q_2 - q_1 - 2$ . As igualdades  $\lceil \log \frac{i}{3} \rceil = q_1$  e  $\lceil \log \frac{i+1}{3} \rceil = q_1 + 1$  só ocorrem quando  $i = 3 \cdot 2^{q_1}$ . Além disso, as igualdades  $\lceil \log \frac{n-i}{3} \rceil = q_2$  e  $\lceil \log \frac{n-i-1}{3} \rceil = q_2 - 1$  só ocorrem quando  $n-i = 3 \cdot 2^{q_2-1} + 1$ . Então  $f(i) - f(i+1) = q_2 - q_1 - 1 \geq 0$ , uma vez que, aqui, temos  $q_2 > q_1 + 1$ .

Em todos os casos possíveis,  $f(i) \geq f(i+1)$  para  $i \in [3, \lfloor \frac{n}{2} \rfloor - 1]$ . Consequentemente, o valor mínimo para  $f$  se dá quando  $i = \lfloor \frac{n}{2} \rfloor$ , e a árvore par com  $\lfloor \frac{n}{2} \rfloor$  símbolos na subárvore esquerda é ótima e equilibrada. Portanto, a hipótese também é válida para  $n$  símbolos e a indução se completa. ■

## 4.2.2 Caracterização de árvores pares ótimas

Na subseção anterior, definimos árvores pares equilibradas e mostramos que elas são ótimas. Entretanto, há outras árvores pares ótimas que não são equilibradas, conforme mostrado através da Tabela 4.1. Nesta seção caracterizamos todas as árvores pares ótimas.

Seja  $T$  uma árvore binária. Denotemos por  $L_T$  o número de folhas de  $T$ .

**Teorema 4.2.4** *Uma árvore par para  $n > 5$  símbolos é ótima se e somente se*

(i)  $T_L$  é uma árvore par ótima e  $T_R$  uma árvore ímpar ótima, e

(ii)  $L_{T_L}$  ou  $L_{T_R}$  é um inteiro no intervalo  $[i_{min}, \lfloor n/2 \rfloor]$ , onde

$$i_{min} = \begin{cases} 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1} & \text{se } n \leq 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1} \\ n - 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor} & \text{caso contrário} \end{cases}$$

**Demonstração:** Seja  $T$  uma árvore par ótima para  $n$  símbolos. Mostraremos que  $T$  satisfaz (i) e (ii). Se (i) não for satisfeito, então substituindo a subárvore esquerda ou direita de  $T$ , respectivamente, por uma árvore par ótima ou uma árvore ímpar ótima com o mesmo número de símbolos, conseguimos reduzir o custo da árvore, uma contradição.

Agora provaremos que a condição (ii) também é satisfeita, usando o Teorema 4.2.3. Sem perda de generalidade, suponhamos  $L_{T_L} \leq L_{T_R}$ . Podemos escrever  $n = 3 \cdot 2^{q-1} + p$ ,  $p$  e  $q$  inteiros,  $1 \leq p \leq 3 \cdot 2^{q-1}$ . Chamemos  $\lceil \log \frac{i}{3} \rceil = q_1$ ,  $\lceil \log \frac{n-i}{3} \rceil = q_2$  e  $\lceil \log \frac{n}{3} \rceil = q$ . A função  $f(i) = P_T(n)$  é não decrescente no intervalo  $[3, \lfloor \frac{n}{2} \rfloor]$  e tem um mínimo para  $i = \lfloor \frac{n}{2} \rfloor$ . Consequentemente, há exatamente um valor mínimo  $i_{min}$  para o qual  $f(i)$  permanece constante no intervalo  $[i_{min}, \lfloor \frac{n}{2} \rfloor]$ . Para cada valor  $i$  deste intervalo temos uma ou mais árvores distintas ótimas  $T$  tal que  $T_L$  tenha  $i$  folhas de codificação. Determinemos  $i_{min}$ . Há dois valores iniciais possíveis, a partir dos quais  $f(i)$  é constante, dependendo de certas condições envolvendo  $q_1$  e  $q_2$ .

*Caso 1:*  $q_1 = q_2$ . Então  $\lceil \log \frac{i_{min}}{3} \rceil = q_1 = \lceil \log \frac{\lfloor \frac{n}{2} \rfloor}{3} \rceil = \lceil \log \frac{n-i}{3} \rceil = q_2$ . Agora temos dois subcasos:



*Subcaso 1.1:*  $n$  é da forma  $n = 3 \cdot 2^{q-1} + 1$ . Portanto,  $q_1 = q_2 = q - 2$ . Neste caso, há apenas um valor possível para  $i_{min}$ :

$$i_{min} = \lfloor n/2 \rfloor = 3 \cdot 2^{q-2}. \quad (4.1)$$

*Subcaso 1.2:*  $n$  não é da forma  $n = 3 \cdot 2^{q-1} + 1$ . Portanto,  $q_1 = q_2 = q - 1$ , e  $i_{min}$  deve satisfazer duas desigualdades:

$$3 \cdot 2^{q-2} + 1 \leq i_{min} \leq \lfloor n/2 \rfloor \quad (4.2)$$

$$n - i_{min} \leq 3 \cdot 2^{q-1} \quad (4.3)$$

De (4.2) e (4.3), temos:

$$\max\{3 \cdot 2^{q-2} + 1, n - 3 \cdot 2^{q-1}\} \leq i_{min} \leq \lfloor n/2 \rfloor \quad (4.4)$$

*Caso 2:*  $q_2 = q_1 + 1$ . Neste caso,  $i_{min}$  é da forma  $i_{min} = 3 \cdot 2^{q_1}$ . Temos dois subcasos:

*Subcaso 2.1:*  $n$  é da forma  $n = 3 \cdot 2^q$ . Vamos examinar os possíveis valores para  $q_1$ . Sabemos que  $q_1 < q$ . Não podemos ter  $q_1 = q - 1$  porque isso implicaria  $n - i_{min} = 3 \cdot 2^{q-1}$  e  $\lceil \log \frac{n - i_{min}}{3} \rceil = q_2 = q - 1 = q_1$ . Também não podemos ter  $q_1 < q - 1$  porque isso implicaria  $n - i_{min} > 3 \cdot 2^{q-1}$  e  $\lceil \log \frac{n - i_{min}}{3} \rceil = q_2 = q$ . Portanto,  $q_2 > q_1 + 1$ . Então não temos valores possíveis para  $q_1$ . Consequentemente,  $n$  não pode ser da forma  $n = 3 \cdot 2^q$ .

*Subcaso 2.2:*  $n$  é da forma  $n = 3 \cdot 2^{q-1} + p$ ,  $1 \leq p < 3 \cdot 2^{q-1}$ . Vamos checar os possíveis valores para  $q_1$ . Não podemos ter  $q_1 = q$  pois isso implicaria  $n - 3 \cdot 2^{q-1} = p$  e  $\lceil \log \frac{p}{3} \rceil = q_2 < q - 1 = q_1$ . Além disso, não podemos ter  $q_1 < q - 2$ , pois isso implicaria  $q_2 \geq q - 1$  e, consequentemente,  $q_2 > q_1 + 1$ . Mas podemos ter  $q_1 = q - 2$ , que é o único valor possível para  $q_1$ . Neste caso,  $n - i_{min} = 3 \cdot 2^{q-2} + p$  e  $q_2 = \lceil \log \frac{3 \cdot 2^{q-2} + p}{3} \rceil = q - 1 = q_1 + 1$ . Consequentemente, podemos afirmar as duas relações seguintes:

$$3 \cdot 2^{q-2} = i_{\min} < \lfloor n/2 \rfloor \quad (4.5)$$

$$n - i_{\min} \leq 3 \cdot 2^{q-1} \quad (4.6)$$

De (4.5) e (4.6), temos:

$$n \leq 9 \cdot 2^{q-2} \quad (4.7)$$

Determinamos as condições que  $i_{\min}$  deve satisfazer para a existência de uma árvore par ótima com  $i_{\min}$  folhas de codificação na subárvore esquerda. Essas condições levam à determinação dos valores exatos para  $i_{\min}$ , conforme se segue. Agora, considere os possíveis valores para  $n$ :

*Caso I:*  $n = 3 \cdot 2^q$ . Então  $n = 6 \cdot 2^{q-1} < 9 \cdot 2^{q-1} = 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ . Esta situação corresponde ao Subcaso 1.2. Aplicando 4.4,  $i_{\min} = 3 \cdot 2^{q-1} = \lfloor n/2 \rfloor$ . Então, há um único elemento no intervalo. Podemos escrever  $i_{\min} = 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ , o que corresponde à primeira situação do teorema.

*Caso II:*  $n = 3 \cdot 2^{q-1} + 1$ . Então  $n = 6 \cdot 2^{q-1} + 1 < 9 \cdot 2^{q-1} = 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ . Esta situação corresponde ao Subcaso 1.1. Aplicando 4.1,  $i_{\min} = 3 \cdot 2^{q-2}$ . Então também só há um elemento no intervalo. Podemos escrever  $i_{\min} = 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ , que também corresponde à primeira situação do teorema.

*Caso III:*  $n \leq 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$  e Casos I e II não ocorrem. Isto é equivalente a dizer que  $n$  é da forma  $n = 3 \cdot 2^{q-1} + p$ ,  $2 \leq p \leq 3 \cdot 2^{q-2}$ . O menor valor para  $i_{\min}$  ocorre quando aplicamos (4.5), porque, se aplicássemos (4.4) teríamos um valor pior para  $i_{\min} \geq 3 \cdot 2^{q-2} + 1$ . Podemos reescrever esta condição como  $i_{\min} = 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ , pois aqui temos  $\lfloor \log \frac{n}{3} \rfloor = q - 1 = \lceil \log \frac{n}{3} \rceil - 1$ . Este caso conclui a primeira situação do teorema.

*Caso IV:*  $n > 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$  e Casos I, II, e III não ocorrem. Nesta situação, devemos ter  $n = 3 \cdot 2^{q-1} + p$ ,  $3 \cdot 2^{q-2} < p < 3 \cdot 2^{q-1}$ , pois  $9 \cdot 2^{q-2} = 3 \cdot 2^{q-1} + 3 \cdot 2^{q-2}$

implica  $\lfloor \log \frac{n}{3} \rfloor = q - 1 = \lceil \log \frac{n}{3} \rceil - 1$ . Portanto, isto é equivalente a  $n > 9 \cdot 2^{\lceil \log \frac{n}{3} \rceil - 1}$ . Só há uma maneira de satisfazer 4.4 no Caso 1:  $\lfloor \frac{n}{2} \rfloor \geq i_{min} \geq \max\{3 \cdot 2^{q-2} + 1, n - 3 \cdot 2^{q-1}\}$ . Mas temos  $n - 3 \cdot 2^{q-1} > 9 \cdot 2^{q-2} - 3 \cdot 2^{q-1} = 3 \cdot 2^{q-2}$ . Então  $i_{min} = n - 3 \cdot 2^{q-1}$ , ou seja,  $i_{min} = n - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}$ , correspondendo à segunda situação do teorema. Isto completa a prova de que a condição é necessária.

Para a suficiência da condição, suponhamos que  $T$  seja uma árvore par satisfazendo (i) e (ii). A prova da necessidade e o Teorema 4.2.3 implicam que uma árvore par ótima satisfaz (i) e (ii). Além disso, existe uma árvore par ótima  $T'$  satisfazendo (i), (ii) e também com  $L_{T'} = L_T$ . Consequentemente,  $T'$  e  $T$  têm mesmo custo, o que significa que  $T$  é de fato ótima. ■

O teorema acima permite o reconhecimento direto de árvores pares ótimas. Também provê um método para gerar todas as árvores pares ótimas para dado conjunto de símbolos.

Podemos ilustrar o teorema 4.2.4 apresentando a tabela 4.2, para uma faixa de valores de 1 a 30, que mostra todas as configurações ótimas de número de codificações nas subárvores esquerda/direita de uma árvore par ótima, para cada  $n$ .

n	Configurações ótimas	n	Configurações ótimas	n	Configurações ótimas
1	0/0	11	5/6	21	9/12 10/11
2	1/1	12	6/6	22	10/12 11/11
3	1/2	13	6/7	23	11/12
4	1/3 2/2	14	6/8 7/7	24	12/12
5	2/3	15	6/9 7/8	25	12/13
6	3/3	16	6/10 7/9 8/8	26	12/14 13/13
7	3/4	17	6/11 7/10 8/9	27	12/15 13/14
8	3/5 4/4	18	6/12 7/11 8/10 9/9	28	12/16 13/15 14/14
9	3/6 4/5	19	7/12 8/11 9/10	29	12/17 13/16 14/15
10	4/6 5/5	20	8/12 9/11 10/10	30	12/18 13/17 14/16 15/15

Tabela 4.2: Configurações ótimas para árvores pares, probabilidades uniformes

Os seguintes corolários podem ser confirmados a partir da tabela 4.2.

**Corolário 4.2.5** *Só existe uma árvore par ótima quando o número de símbolos é da forma  $n = 3 \cdot 2^q$ ,  $q \geq 0$ , inteiro.*

**Corolário 4.2.6** *Toda árvore par ótima, é obtida pela composição das subárvores ótimas para 2, 3 ou 4 símbolos mostradas na Figura 4.2.*

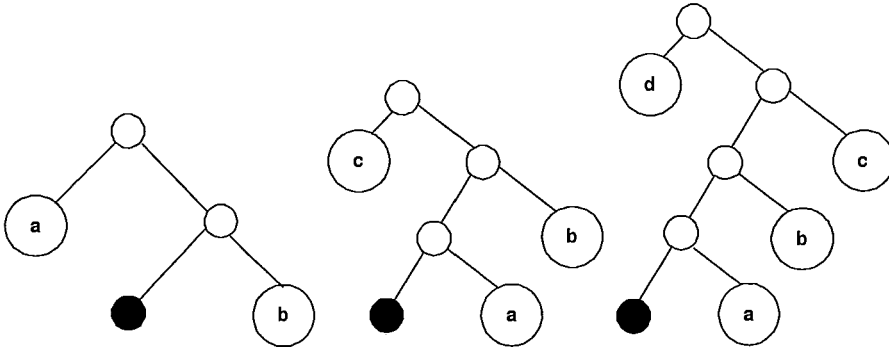


Figura 4.2: Árvores pares ótimas para 2, 3 e 4 símbolos

**Demonstração:** Para  $n = 2, 3$  ou  $4$ , o enunciado é óbvio. Para  $n = 5$ , podemos verificar, por tentativa, que há apenas 2 árvores ótimas distintas, composição dos dois primeiros tipos de árvore. Para  $n > 5$ , o número mínimo de nós na subárvore esquerda é sempre maior que 1 pois, considerando os dois valores mínimos possíveis, temos:

a)  $3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1} \geq 3 \cdot 2^0 = 3 > 1$ , vale sempre, inclusive se  $n \leq 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ .

b)  $n - 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor} > 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1} - 3 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor} = (3/2) \cdot 2^{\lfloor \log \frac{n}{3} \rfloor} > 1$ , se  $n > 9 \cdot 2^{\lfloor \log \frac{n}{3} \rfloor - 1}$ .

Aplicando-se, recursivamente, o raciocínio às duas subárvores esquerda e direita, chega-se sempre a uma subárvore ótima contendo 2 a 4 folhas, que tem que ser, obrigatoriamente, uma das mencionadas. ■

Para completar a caracterização de árvores pares ótimas, vamos determinar as profundidades mínimas e máximas os números mínimo e máximo de folhas de erro e das árvores pares ótimas.

É bastante intuitivo que, dados  $n$  símbolos, as árvores pares ótimas de profundidades mínima e máxima podem ser obtidas a partir da seguinte idéia:

a) As árvores pares ótimas de profundidade mínima são construídas, recursivamente, utilizando subárvores à esquerda com o maior número de folhas de codificação, desde que não superior ao da direita. Portanto, essas subárvores têm  $\lfloor \frac{n}{2} \rfloor$  folhas de codificação. Conforme apresentado anteriormente, estas são as árvores pares equilibradas.

b) As árvores pares ótimas de profundidade máxima são construídas, também de forma recursiva, utilizando subárvores à esquerda com o número mínimo de folhas de codificação, dado pelo teorema 4.2.4.

Sejam  $D_{min}(n)$  e  $D_{max}(n)$  as profundidades mínima e máxima, respectivamente, de uma árvore par ótima para  $n$  símbolos. Temos:

$$D_{min} = \begin{cases} 1, & \text{se } n = 1 \\ 1 + D_{min}(\lfloor n/2 \rfloor), & \text{se } n > 1 \end{cases}$$

$$D_{max} = \begin{cases} 1, & \text{se } n = 1 \\ 1 + D_{max}(n - i_{min}), & \text{se } n > 1. \end{cases}$$

Resolvendo essas recorrências, obtemos:

$$D_{min} = \begin{cases} 1, & \text{se } n = 1, \\ \lceil \log n \rceil + 1, & \text{se } n > 1 \end{cases}$$

$$D_{max} = \begin{cases} 1, & \text{se } n = 1 \\ \lceil \log \frac{n}{3} \rceil + 3, & \text{se } n > 1. \end{cases}$$

A tabela 4.3 foi construída usando as recorrências acima e os dados da Tabela 3.1, para números de símbolos,  $n$ , de 1 a 60.

Dados  $n$  símbolos, para encontrar os números mínimo,  $E_{min}(n)$  e máximo,  $E_{max}(n)$  de folhas de erro nas árvores pares ótimas, podemos usar uma es-

n	$D_{min}$	$D_{max}$	n	$D_{min}$	$D_{max}$	n	$D_{min}$	$D_{max}$	n	$D_{min}$	$D_{max}$
1	0	0	16	5	6	31	6	7	46	7	7
2	2	2	17	6	6	32	6	7	47	7	7
3	3	3	18	6	6	33	7	7	48	7	7
4	3	4	19	6	6	34	7	7	49	7	8
5	4	4	20	6	6	35	7	7	50	7	8
6	4	4	21	6	6	36	7	7	51	7	8
7	4	5	22	6	6	37	7	7	52	7	8
8	4	5	23	6	6	38	7	7	53	7	8
9	5	5	24	6	6	39	7	7	54	7	8
10	5	5	25	6	7	40	7	7	55	7	8
11	5	5	26	6	7	41	7	7	56	7	8
12	5	5	27	6	7	42	7	7	57	7	8
13	5	6	28	6	7	43	7	7	58	7	8
14	5	6	29	6	7	44	7	7	59	7	8
15	5	6	30	6	7	45	7	7	60	7	8

Tabela 4.3: Profundidades mínima e máxima para árvores pares ótimas

tratégia análoga à do algoritmo da Seção 4.2.1, que encontra o caminho de codificação mínimo  $P(n)$ .

A idéia é verificar, em todas as configurações de árvores ótimas, com  $j$  símbolos na subárvore esquerda e  $n - j$  na direita, qual o valor mínimo para a soma  $E_{min}(j) + E_{min}(n - j)$ , que será o  $E_{min}(n)$  procurado e, de forma análoga, qual o valor máximo para  $E_{max}(j) + E_{max}(n - j)$ , que será  $E_{max}(n)$ .

As recorrências podem ser expressas da seguinte maneira:

$$E_{min} = \begin{cases} 0, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ \min(E_{min}(j) + E_{min}(n - j)), & \text{se } n > 2 \text{ e } P(n) = P(j) + P(n - j) + n, \\ & 1 \leq j \leq (n - 2). \end{cases}$$

$$E_{max} = \begin{cases} 0, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ \max(E_{max}(j) + E_{max}(n - j)), & \text{se } n > 2 \text{ e } P(n) = P(j) + P(n - j) + n, \\ & 1 \leq j \leq (n - 2). \end{cases}$$

A tabela 4.4 foi construída usando as recorrências acima, para números de símbolos,  $n$ , de 1 a 60.

n	$E_{min}$	$E_{max}$	n	$E_{min}$	$E_{max}$	n	$E_{min}$	$E_{max}$	n	$E_{min}$	$E_{max}$
1	0	0	16	4	8	31	8	15	46	15	16
2	1	1	17	5	8	32	8	16	47	16	16
3	1	1	18	5	8	33	9	16	48	16	16
4	1	2	19	6	8	34	9	16	49	16	17
5	2	2	20	6	8	35	10	16	50	16	18
6	2	2	21	7	8	36	10	16	51	16	19
7	2	3	22	7	8	37	11	16	52	16	20
8	2	4	23	8	8	38	11	16	53	16	21
9	3	4	24	8	8	39	12	16	54	16	22
10	3	4	25	8	9	40	12	16	55	16	23
11	4	4	26	8	10	41	13	16	56	16	24
12	4	4	27	8	11	42	13	16	57	16	25
13	4	5	28	8	12	43	14	16	58	16	26
14	4	6	29	8	13	44	14	16	59	16	27
15	4	7	30	8	14	45	15	16	60	16	28

Tabela 4.4: Números mínimos e máximos de folhas de erro em árvores pares ótimas

Podemos observar os seguintes pontos, em relação à Tabela 4.4:

a) Para números da forma  $n = 2^k$ ,  $n > 2$ , temos  $E_{min}(n) = 2^{k-2}$  e  $E_{max}(n) = 2^{k-1}$ . Isso pode ser explicado pelo fato de que, para o caso mínimo, as árvores pares ótimas são construídas pela composição unicamente da subárvore ótima para 4 símbolos mostrada na Figura 4.2. Neste caso, o número de folhas de erro é 1/4 do número de folhas de codificação. Para o caso máximo, as árvores pares ótimas são construídas pela composição unicamente da subárvore ótima para 2 símbolos mostrada na Figura 4.2. Neste caso, o número de folhas de erro é metade do número de folhas de codificação.

b) Para números da forma  $n = 3 \cdot 2^k$ ,  $n > 2$ , temos  $E_{min}(n) = E_{max}(n) = 2^k$ . Isso pode ser explicado pelo fato de que, neste caso, há apenas uma configuração ótima, sendo que a árvore é construída pela composição unicamente da subárvore ótima para 3 símbolos mostrada na Figura 4.2. Aqui, o número de folhas de erro é 1/3 do número de folhas de codificação.

c) Conjectura-se que as soluções das recorrências sejam as seguintes:

$$E_{min} = \begin{cases} 0, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ \min(2^{\lceil \log \frac{n}{2} \rceil - 1}, \lfloor \frac{n}{2} \rfloor - 2^{\lceil \log \frac{n}{2} \rceil - 2}), & \text{se } n > 2. \end{cases}$$

$$E_{max} = \begin{cases} 0, & \text{se } n = 1 \\ 1, & \text{se } n = 2 \\ \max(2^{\lceil \log n \rceil - 2}, n - 2^{\lceil \log n \rceil - 1}), & \text{se } n > 2. \end{cases}$$

### 4.2.3 Comparação de árvores ótimas com árvores de Huffman

Para finalizar o estudo sobre árvores pares ótimas para probabilidades uniformes vamos comparar profundidades e custos dessas árvores com as correspondentes árvores de Huffman.

Nas árvores de Huffman, as profundidades mínima e máxima coincidem. Seja  $H$  uma árvore de Huffman para  $n$  símbolos e  $D_H$  sua profundidade. Então,



$D_H = \lceil \log n \rceil$ . Comparando  $D_{min}$  com  $D_H$ , concluímos que sempre podemos construir uma árvore par ótima cuja codificação mais longa seja um bit maior que a da correspondente na árvore de Huffman. Dos valores de  $D_{max}$  e  $D_H$ , podemos adicionalmente concluir que a codificação de maior tamanho em uma árvore par ótima qualquer tem, no máximo, 2 bits a mais do que a da maior codificação na árvore de Huffman.

A seguir, consideramos custo, que equivale a tamanho médio de codificações.

**Teorema 4.2.7** *A diferença de custo entre uma árvore par ótima e uma árvore de Huffman, para  $n > 1$  símbolos, varia no intervalo  $[1/3, 1/2]$ . É mínima quando  $n = 3 \cdot 2^k$ , e máxima quando  $n = 2^k$ , para algum  $k$ , inteiro.*

**Demonstração:** O custo de uma árvore par ótima (4.2.4) é:

$$\frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}}{n}$$

enquanto o de uma árvore de Huffman [18] é:

$$\frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

Então, a diferença é dada por:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil} - (n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil})}{n}$$

Primeiro, vamos provar que  $d \leq 1/2$ . Consideremos os dois casos seguintes:

*Caso 1:*  $n = 2^k$ ,  $k \geq 1$ . Então  $\lceil \log n \rceil = k$  and  $\lceil \log \frac{n}{3} \rceil = \lceil \log ((2^{k-2})(\frac{4}{3})) \rceil = \lceil (k-2) + \log \frac{4}{3} \rceil = k-2+1 = k-1$ . Concluímos que:

$$d = \frac{2^k(k-1+3) - 3 \cdot 2^{k-1} - (2^k(k+1) - 2^k)}{2^k} = 1/2$$

*Caso 2:*  $n = 2^k + p$ ,  $1 < p < 2^k$ . Então  $\lceil \log n \rceil = k + 1$  e  $\lceil \log \frac{n}{3} \rceil$  pode ser igual a  $k - 1$  ou a  $k$ . Temos dois subcasos:

*Subcaso 2.1:* se  $\lceil \log \frac{n}{3} \rceil = k - 1$  então  $d = \frac{2^{k-1}}{2^k + p}$ . Usando o fato de que  $p > 1$ , concluímos que  $d < 1/2$ .

*Subcaso 2.2:* se  $\lceil \log \frac{n}{3} \rceil = k$  então  $d = \frac{p}{2^k + p}$ . Usando o fato de que  $p < 2^k$ , concluímos que  $d < 1/2$ .

Portanto, a diferença máxima entre as fuas codificações é  $1/2$ . Em seguida, vamos mostrar que  $d$  é, no mínimo,  $1/3$ . Consideremos dois casos adicionais:

*Caso 3:*  $n = 3 \cdot 2^k$ ,  $k \geq 0$ . Então  $\lceil \log n \rceil = \lceil k + \log 3 \rceil = (k + 2)$  e  $\lceil \log \frac{n}{3} \rceil = \lceil \log 2^k \rceil = k$ . Consequentemente:

$$d = \frac{-3 \cdot 2^k + 2^{k+2}}{3 \cdot 2^k} = 1/3$$

*Caso 4:*  $n = 3 \cdot 2^k + p$ ,  $1 < p < 3 \cdot 2^k$ . Então  $\lceil \log n \rceil$  pode ser igual a  $(k + 2)$  ou a  $(k + 3)$ , e  $\lceil \log \frac{n}{3} \rceil = k + 1$ . Consideremos dois subcasos adicionais.

*Subcaso 4.1:* se  $\lceil \log n \rceil = k + 2$  então  $d = \frac{2^k + p}{3 \cdot 2^k + p}$ . Uma vez que  $p > 1$ , concluímos que  $d > 1/3$ .

*Subcaso 4.2:* se  $\lceil \log n \rceil = k + 3$  então  $d = \frac{2 \cdot 2^k}{3 \cdot 2^k + p}$ . Como temos  $p < 3 \cdot 2^k$ , concluímos que  $d > 1/3$ .

Portanto, nos dois subcasos acima, a diferença mínima entre as codificações é  $1/3$ . Isso completa a prova. ■

Os limites  $1/3$  e  $1/2$  podem ser explicados em termos das árvores básicas da Figura 4.2, conforme discutido a seguir.

Quando  $n$  é da forma  $n = 2^k$ , a árvore de Huffman é uma árvore binária cheia, com todas as folhas tendo mesma profundidade  $\lceil \log n \rceil$ . Por outro lado, a árvore par ótima equilibrada é composta de  $2^{k-1}$  subárvores ótimas com 2

codificações. Nesta árvore, metade das folhas de codificação tem profundidade  $\lceil \log n \rceil$  e a outra metade tem profundidade  $\lceil \log n \rceil + 1$ . Portanto, a diferença de caminhos de codificação é  $n/2$ , resultando numa diferença de custos de  $1/2$ .

Quando  $n$  é da forma  $n = 3 \cdot 2^k$ , a árvore de Huffman é uma árvore binária completa, tendo  $2^{k+1}$  folhas com profundidade  $\lceil \log n \rceil$ , e  $2^k$  folhas com profundidade  $\lceil \log n \rceil - 1$ . Por outro lado, só há um tipo de árvore par ótima, que é composta de  $2^k$  subárvores ótimas para três símbolos. Neste caso, temos  $2^k$  folhas de codificação com profundidade  $\lceil \log n \rceil + 1$ ,  $2^k$  folhas de codificação com profundidade  $\lceil \log n \rceil$ , e  $2^k$  folhas de codificação com profundidade  $\lceil \log n \rceil - 1$ . A diferença de caminhos de codificação é  $2^k = n/3$ , resultando numa diferença de custos de  $1/3$ .

Nos demais casos para  $n$ , as árvores equilibradas têm configuração intermediária entre esses limites.

A tabela 4.5 ilustra as diferenças de caminho de codificação e de custo.

n	árvores pares ótimas		árvores de Huffman		d
	P(n)	c(n)	P(n)	c(n)	
10	38	3.8	34	3.4	0.4
24	120	5	112	4.67	0.33
32	176	5.5	160	5	0.5
48	288	6	272	5.67	0.33
64	416	6.5	384	6	0.5
100	708	7.08	672	6.72	0.36
192	1536	8	1472	7.67	0.33
500	4732	9.46	4488	8.98	0.48
1000	10464	10.46	9976	9.98	0.48
5000	63856	12.77	61808	12.36	0.41
10000	137712	13.77	133616	13.36	0.41

Tabela 4.5: Diferenças de custo entre árvores pares ótimas e árvores de Huffman

Observando a tabela, verificamos que os valores da forma  $n = 3 \cdot 2^k$  são  $n = 24, 48$  e  $192$ , para os quais  $d = 1/3$ ; os valores da forma  $n = 2^k$  são  $n = 32$  e  $64$ , com  $d = 1/2$ . Os demais valores não são de nenhuma dessas duas formas

e temos, então  $1/3 < d < 1/2$ .

Na próxima seção consideraremos o caso geral de árvores pares ótimas, onde temos probabilidades arbitrárias.

### 4.3 Árvores pares ótimas para probabilidades arbitrárias

Nesta seção descrevemos um algoritmo exato para construir uma árvore par ótima para símbolos que tenham probabilidades arbitrárias.

Seja  $\mathcal{S} = \{s_1, \dots, s_n\}$  um conjunto de símbolos, cada  $s_i$  com probabilidade  $f_i$  satisfazendo  $f_i \leq f_{i+1}$ , para  $1 \leq i < n$ . Para  $m \leq n$ , denotaremos  $\mathcal{S}_m = \{s_1, \dots, s_m\}$ .

Nosso objetivo é encontrar um código ótimo  $\mathcal{C}$  (ou sua correspondente árvore par ótima,  $T$ ) para  $\mathcal{S}$  com mínimo custo. Na verdade, proporemos a solução de um problema um pouco mais geral.

Uma *floresta de paridade*  $F$  para  $\mathcal{S}_m$  é um conjunto de  $q$  árvores pares e  $q$  árvores ímpares, para algum  $q \in \{1, \dots, m\}$ , tal que as folhas de paridade par das árvores pares e as folhas de paridade ímpar das árvores ímpares correspondam aos símbolos de  $\mathcal{S}_m$ .

Definamos o *custo* de  $F$  como a soma dos custos de suas árvores. Dizemos que  $F$  é  $(m, q)$ -ótima quando seu custo é o menor entre todas as florestas para  $\mathcal{S}_m$ , tendo  $q$  árvores pares e  $q$  árvores ímpares. Denotaremos por  $c(m, q)$  o custo de uma floresta  $(m, q)$ -ótima.

Inicialmente, definamos a função

$$A_i = \begin{cases} \sum_{j=1}^i f_j, & \text{se } i > 0 \\ 0, & \text{caso contrário} \end{cases}$$

Em termos desta notação, a solução para o nosso problema é uma árvore tendo como subárvores as árvores de uma floresta  $(n, 1)$ -ótima. Seu custo é  $c(n, 1) + A_n$ .

O seguinte teorema descreve o cálculo de  $c(m, q)$ .

**Teorema 4.3.1** *Sejam  $m, q$  inteiros tais que  $1 \leq m \leq n$  e  $1 \leq q \leq m$ .*

(1) *se  $m \leq q$  então  $c(m, q) = 0$*

(2) *se  $m > q$ , então  $c(m, q) = \min_{0 \leq i \leq q} \{c(m - i, 2q - i) + A_{m-i}\}$ .*

**Demonstração:** Por indução, mostramos que os casos (1)-(2) computam corretamente  $c(m, q)$ , para  $1 \leq m \leq n$  e  $1 \leq q \leq m$ . Quando  $m = 1$ , o caso (1) implica em  $c(1, q) = 0$ , que é correto, uma vez que o único símbolo está na raiz de uma árvore par. Temos também uma raiz ímpar como folha de erro e as demais árvores são nulas. Para  $m > 1$ , seja  $F$  uma floresta  $(m, q)$ -ótima para  $\mathcal{S}_m$ . Consideremos as seguintes alternativas.

(1)  $m \leq q$ : Neste caso,  $m$  árvores pares triviais de  $F$  correspondem aos símbolos  $s_1, \dots, s_m$ , respectivamente. As restantes  $q - m$  árvores pares são nulas. Temos também  $m$  trivial árvores ímpares triviais, correspondentes a folhas de erro e  $q - m$  árvores ímpares nulas. Portanto,  $c(m, q) = 0$ .

(2)  $m > q$ : Neste caso, podemos ter  $i$  árvores triviais de  $F$  correspondendo aos símbolos  $s_{m-i+1}, \dots, s_m$ , para algum  $i \in \{0, \dots, q\}$ . Os restantes  $m - i$  símbolos estão distribuídos numa floresta  $F'$  com  $q - i$  árvores pares e  $q$  árvores ímpares. Note que a floresta  $F''$  formada pela remoção das raízes das árvores de  $F'$  é uma floresta  $(m - i, 2q - i)$ -ótima. Isso implica que o custo de  $F'$  é dado por  $c(m - i, 2q - i) + A_{m-i}$ . Portanto,  $c(m, q)$  deve ser o mínimo de  $c(m - i, 2q - i) + A_{m-i}$ ,  $0 \leq i \leq q$ . ■

O Teorema 4.3.1 conduz a um algoritmo de Programação Dinâmica para determinar  $c(m, q)$ , para todo  $1 \leq m \leq n$  e  $1 \leq q \leq m$ .

Começemos calculando a função  $A_i$ , para  $1 \leq i \leq n$ . O parâmetro  $m$  varia de forma decrescente,  $1 \leq m \leq n$ . O primeiro custo a ser computado é  $c(m, m)$ , que é 0, por (1). Para cada  $m$ , variamos  $q$  de forma decrescente,

$1 \leq q < m$ , e, para cada par  $(m, q)$ , computamos  $c(m, q)$  aplicando (2). O algoritmo pára quando  $c(n, 1)$  é calculado, pois nosso objetivo é obter  $c(n, 1) + A_n$ . Há  $O(n^2)$  subproblemas. A solução de cada subproblema é feita em tempo constante, se usarmos a equação (1), ou em tempo  $O(n)$ , quando avaliado por (2). Consequentemente, a complexidade em tempo é  $O(n^3)$ . Os requisitos de espaço são  $O(n^2)$ .

O algoritmo apresentado foi obtido a partir de uma sugestão dada por A. A. Pessoa [30], tendo melhorado o algoritmo anterior, cuja complexidade era  $O(n^3 \log n)$ .

A Figura 4.3 mostra uma árvore par ótima para 8 símbolos com probabilidades  $\{1/24, 1/24, 1/12, 1/12, 1/8, 1/8, 1/4, 1/4\}$ . Mostra também a árvore de Huffman correspondente. Na representação das árvores indicamos as probabilidades de cada símbolo, ao invés do mesmo. Os custos das árvores são  $34/12$  e  $33/12$ , respectivamente.

Os limites de custo e as comparações entre árvores pares ótimas e árvores de Huffman serão feitos no próximo capítulo.

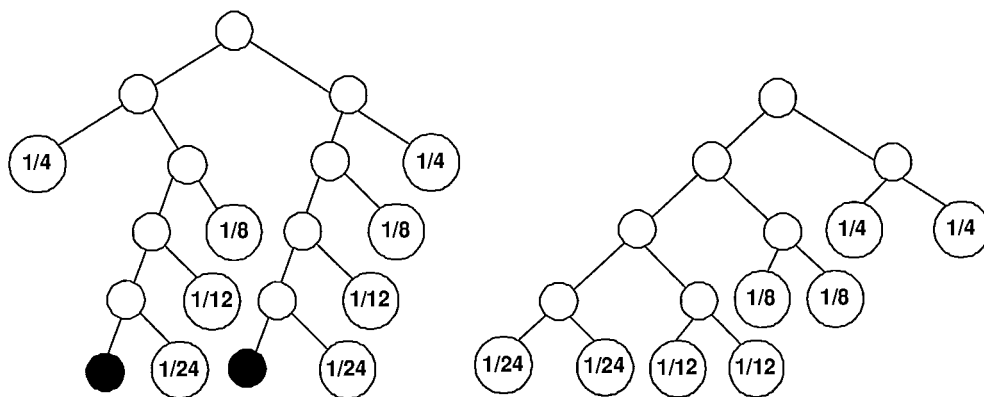


Figura 4.3: Árvore par ótima e árvore de Huffman para 8 símbolos

# Capítulo 5

## Heurísticas para árvores pares

Neste capítulo descrevemos duas heurísticas para obter códigos pares próximos dos códigos pares ótimos. Apresentamos também um limite superior para a diferença de custo entre as árvores pares obtidas através da segunda heurística e as correspondentes árvores de Huffman, que também é um limite para as árvores pares ótimas.

### 5.1 Heurística 1

A Heurística 1 é muito simples e baseada numa leve modificação no algoritmo clássico de Huffman [15].

Dados  $n$  símbolos com probabilidades  $f_1 \leq f_2 \leq \dots \leq f_n$ , a Heurística 1 consiste em dois passos:

**Passo 1.** *Executar o algoritmo de Huffman, para obter a árvore de Huffman  $T_H$  para  $n$  símbolos.*

**Passo 2.** *Converter  $T_H$  em uma árvore par  $T_{U_1}$ , da seguinte forma: para cada folha ímpar  $z$  correspondente ao símbolo  $s_i$ , criar dois filhos  $z_L$  e  $z_R$ , tal que:*

- o filho esquerdo  $z_L$  é uma folha de erro;
- o filho direito  $z_R$  é a nova folha de codificação correspondente a  $s_i$ . Chamamos  $z_R$  uma folha ampliada.

Observe que o tempo de execução da Heurística 1 é  $O(n \log n)$ , dado que é dominada pelo Passo 1. O Passo 2 pode ser realizado facilmente no tempo  $O(n)$ . Em termos de espaço, a necessidade é a mesma do algoritmo de Huffman, que é  $O(n)$ .

A Figura 5.1 ilustra o processo, para o mesmo conjunto de símbolos da Figura 4.3.

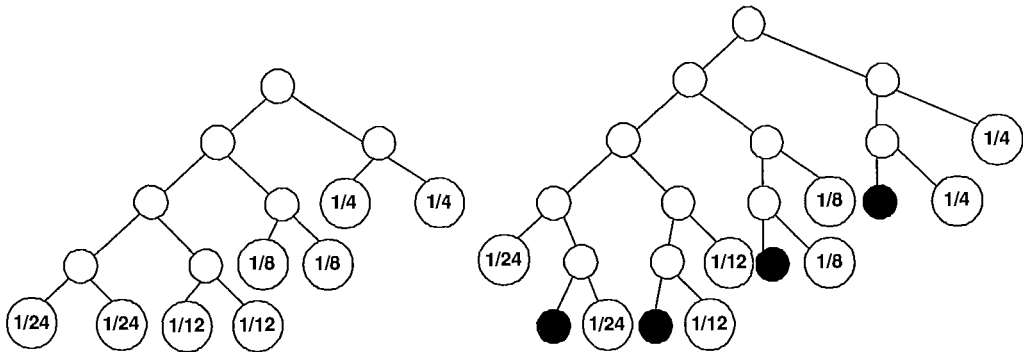


Figura 5.1: Árvore de Huffman e árvore par da Heurística 1

## 5.2 Heurística 2

A Heurística 2 adiciona algumas melhorias à anterior. Como veremos, essas melhorias permitem obter códigos pares muito próximos dos códigos pares ótimos.

Vejamos três maneiras possíveis de melhorar a heurística previamente descrita. Mostraremos que essas melhorias produzem um crescimento qualitativo na performance da geração da árvore par, sem aumentar a sua complexidade.

**Melhoria I.** Durante o Passo 1 (execução do algoritmo de Huffman), acrescentar o seguinte teste:

*Dentre os pares de subárvores candidatos a serem agregados no início de cada iteração, dar preferência ao par de subárvores  $T_1$  e  $T_2$  tal que  $T_1$  é trivial e  $T_2$  não.*



Em outras palavras, a idéia é evitar ao máximo a agregação de subárvores triviais. A razão desta estratégia é explicada a seguir.

Em  $T_H$ , sempre que se juntam duas subárvores triviais elas permanecerão para sempre como folhas irmãs. Garantidamente uma delas será uma folha ímpar, no final do algoritmo. Quando forçamos uma subárvore trivial se agregar a uma subárvore não trivial, nós diminuimos o número de folhas irmãs entre si em  $T_H$ , e, conseqüentemente, o número de folhas “garantidamente ímpares”. Em muitos casos, esta estratégia diminui o custo adicional para produzir uma árvore par, no Passo 2.

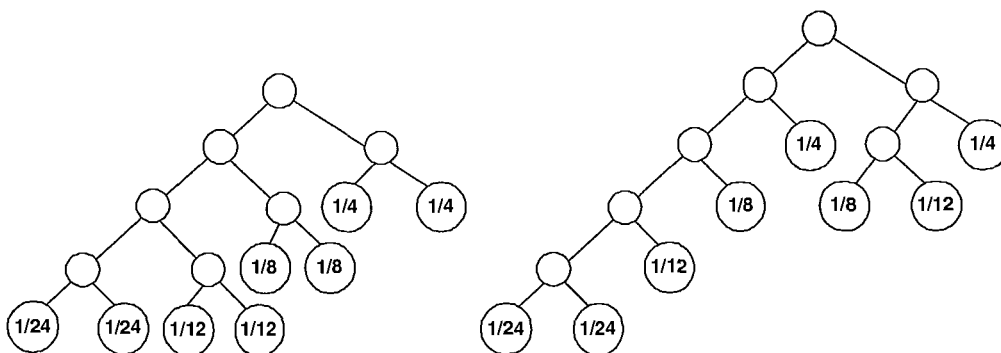


Figura 5.2: Árvore de Huffman antes e após Melhoria I

A Figura 5.2 ilustra o procedimento, considerando o mesmo conjunto de símbolos dos exemplos anteriores. Observe-se que temos duas árvores de Huffman, sendo que a primeira é uma árvore que poderia ser obtida pelo algoritmo de Huffman normal e a segunda, através da introdução da Melhoria I. Enquanto a primeira tem 4 pares de folhas irmãs, a segunda tem somente 2 pares.

Chamemos  $T_{H_1}$  a árvore de Huffman obtida pela Melhoria I. É importante ressaltar que esta melhoria não afeta a essência do algoritmo de Huffman, uma vez que  $T_{H_1}$  é uma árvore plausível.

Além disso, é possível implementar a melhoria I em tempo constante, mantendo dois “heaps”  $H'$  and  $H''$  durante a execução do algoritmo de Huffman, onde os nós de  $H'$  contêm subárvores triviais e os nós de  $H''$ , as restantes. No início do algoritmo,  $H'$  contém  $n$  subárvores triviais e  $H''$  é vazio. Quando

se começa uma nova iteração, simplesmente testamos se as raízes  $H'$  e  $H''$  formam um par candidato à agregação parcial; se isso for verdade, esse par é agregado e colocado no heap  $H''$ .

**Melhoria II.** Mudar  $T_{H_1}$  através da aplicação repetida da seguinte operação, em ordem crescente de profundidade:

*Se há dois nós  $z', z''$  com mesma profundidade em  $T_{H_1}$ , tal que  $z'$  é uma folha ímpar e  $z''$  um nó interno par, trocar as posições de  $z'$  e  $z''$ .*

A Figura 5.3 mostra a aplicação da Melhoria II à árvore obtida após a melhoria I, para o mesmo exemplo anterior. Podemos observar que o número de folhas ímpares baixa de 5 para 2.

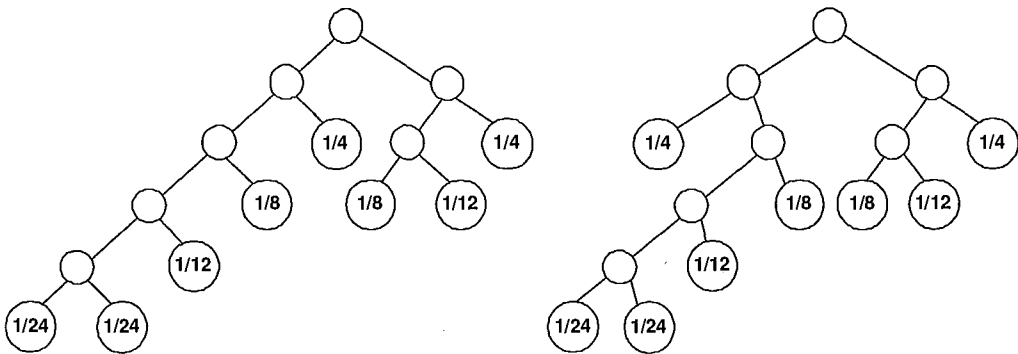


Figura 5.3: Árvore de Huffman antes e após a Melhoria II

Observemos que cada aplicação da operação acima diminui o número de folhas ímpares de  $T_{H_1}$  em uma unidade. Toda vez que encontramos  $k$  folhas ímpares e  $\ell$  folhas internas em algum nível da árvore de profundidade  $i$ , executamos  $\min\{k, \ell\}$  trocas e prosseguimos para a profundidade  $i + 1$ .

É claro que o número de trocas é limitado pelo número de folhas de  $T_{H_1}$ . Como uma única troca realiza mudanças em um número constante de links, a complexidade geral da Melhoria II é  $O(n)$ .

Chamemos  $T_{H_2}$  a árvore de Huffman obtida a partir da Melhoria II. Novamente, a essência do algoritmo de Huffman não é afetada, dado que  $T_{H_2}$  é ainda plausível.

**Melhoria III.** Aplicar o Passo 2 em  $T_{H_2}$ . Seja  $T$  a árvore par obtida.

Então, redistribuir os símbolos entre as folhas de  $T$  como se segue: *Sempre que existirem duas folhas de paridade par  $z_i, z_j$  em  $T$ , com profundidades  $d_i \leq d_j$ , representando os símbolos  $s_i, s_j$  com probabilidades  $f_i \leq f_j$ , respectivamente, então trocar os símbolos atribuídos a  $z_i$  e  $z_j$ .*

Deve ser notado que cada troca realizada na etapa final da Melhoria III reduz o custo da árvore par resultante em  $(d_j - d_i)(f_j - f_i)$ .

A Figura 5.4 mostra a aplicação da melhoria III, à árvore da direita na Figura 5.3, em duas etapas: na esquerda, a obtenção da árvore par e na direita, o ajuste de símbolos. A árvore resultante tem custo  $69/24$ , ao passo que a árvore par ótima tem custo  $68/24$  e a árvore de Huffman,  $66/24$ .

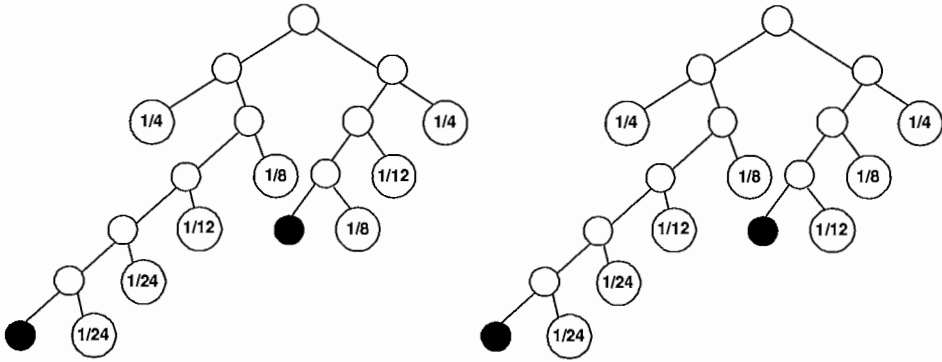


Figura 5.4: Árvore par após cada etapa da Melhoria III

Todo o processo pode ser implementado da seguinte maneira: após a aplicação do Passo 2, ordenar as folhas pares  $z_1, z_2, \dots, z_n$  de  $T$  de acordo com suas respectivas profundidades  $d_1, d_2, \dots, d_n$  usando “bucket sort”. Então, reatribuir as folhas aos símbolos, tal que a folha  $z_i$ , com profundidade  $d_i$ , seja atribuída ao símbolo  $s_{n-i+1}$  com probabilidade  $f_{n-i+1}$ . (lembrar que  $f_1 \leq f_2 \leq \dots, f_n$ ). Com este procedimento, são restauradas possíveis distorções introduzidas pela ampliação das folhas. O tempo requerido para esta operação é, portanto,  $O(n)$ . Conseqüentemente, o tempo total da Heurística 2 é limitado por  $O(n \log n)$ , com  $O(n)$  em espaço.

Outras melhorias possíveis foram inicialmente consideradas, mas depois descartadas, pois os experimentos realizados não mostraram vantagens rele-

vantes. Uma dessas melhorias marginais seria a troca de uma folha de erro com uma subárvore cuja raiz tivesse profundidade superior à dessa folha.

### 5.3 Limites de custo

Nesta seção apresentamos um limite superior analítico para o custo da árvore par gerada pela Heurística 2, relativamente ao custo da árvore de Huffman correspondente.

A terminologia empregada nesta seção é a seguinte: dados  $n$  símbolos com probabilidades  $f_1, f_2, \dots, f_n$ ,  $T_H$  é a árvore de Huffman para esses símbolos;  $T_E$  é a árvore par ótima correspondente;  $T_{U_1}$  é a árvore par obtida pela Heurística 1 e  $T_{U_2}$  é a árvore par obtida pela Heurística 2. Observemos que

$$c(T_H) \leq c(T_E) \leq c(T_{U_2}) \leq c(T_{U_1}).$$

**Lema 5.3.1**  $c(T_{U_2}) \leq c(T_H) + \frac{\sum_{i=1}^n f_i}{2}$ .

**Demonstração:** Este limite é devido às melhorias II e III. Sejam  $n_{ol}(k), n_{oi}(k), n_{el}(k), n_{ei}(k)$  os números de folhas ímpares, nós internos ímpares, folhas pares e nós internos pares, respectivamente, com profundidade  $k$  em  $T_{H_2}$ .

$$\text{Então ou } n_{ol}(k) = 0 \text{ ou } n_{ei}(k) = 0. \quad (1)$$

$$\text{Além disso, é claro que } n_{ol}(k) + n_{oi}(k) = n_{el}(k) + n_{ei}(k). \quad (2)$$

Temos que ter  $n_{ol}(k) \leq n_{el}(k)$  pois, caso contrário, se  $n_{ol}(k) > n_{el}(k)$ , então  $n_{ol}(k) > 0$ , o que implica em  $n_{ei}(k) = 0$ , por (1). Mas, neste caso,  $n_{ol}(k) + n_{oi}(k) = n_{el}(k)$  por (2), ou seja,  $n_{ol}(k) \leq n_{el}(k)$ , uma contradição.

Somando-se  $n_{ol}(k)$  e  $n_{el}(k)$  para todos os valores de  $k$ , concluímos que o número de folhas ímpares é menor ou igual ao número de folhas pares em  $T_{H_2}$ . Portanto, o número de folhas ímpares é, no máximo,  $\lfloor \frac{n}{2} \rfloor$  e é menor ou igual ao número de operações de agregação entre duas subárvores triviais no algoritmo de Huffman. A seguir, o Passo 2 aumenta em um a profundidade das folhas ímpares, para converter  $T_{H_2}$  em uma árvore par.

Agora, quando aplicamos a Melhoria III, as probabilidades são redistribuídas na árvore, de tal forma que qualquer par de folhas  $z_i, z_j$  com profundidades  $d_i, d_j$  e probabilidades  $f_i, f_j$  satisfaça à condição  $f_i \geq f_j \Leftrightarrow d_i \leq d_j$ . Consequentemente, existe uma correspondência um a um entre o conjunto de folhas ampliadas e um subconjunto de folhas pares tal que se  $z_i$  é uma folha ampliada e  $z_j$  sua correspondente folha par, então  $f_i \geq f_j$ . Portanto:

$$c(T_{U_2}) \leq c(T_H) + \frac{\sum_{i=1}^n f_i}{2}. \quad \blacksquare$$

O Lema 7.3.1 afirma que a maior diferença de custos entre uma árvore par obtida pela Heurística 2 e a árvore de Huffman correspondente é, no máximo, 0,5. O próximo teorema fornece outro limite para essa diferença, que pode ser mais justo para distribuições com baixa entropia, onde o custo da árvore de Huffman seja inferior a 3.

**Teorema 5.3.2**  $c(T_{U_2}) \leq \frac{7}{6}c(T_H)$ , se  $n > 4$ .

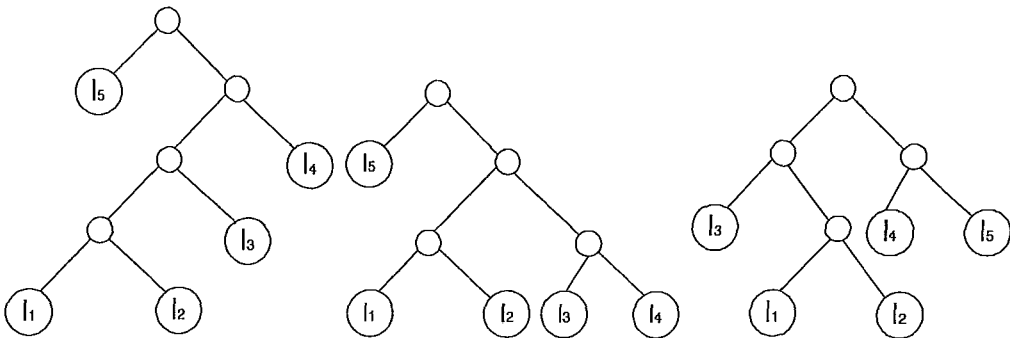


Figura 5.5: Árvores de Huffman para  $n = 5$

**Demonstração:** Nesta prova,  $T_H$  é a árvore de Huffman criada pelo processo definido pela Heurística 2, e  $T_{U_2}$  a correspondente árvore par. Suponhamos, inicialmente,  $n = 5$ . As três únicas formas possíveis para  $T_H$  estão ilustradas na Figura 5.5. Sejam  $f_1 \leq f_2 \leq f_3 \leq f_4 \leq f_5$  as probabilidades das folhas  $l_1, \dots, l_5$ , respectivamente. Consideremos cada uma dessas árvores.

*Caso 1.* Seja  $T_H$  a árvore de Huffman da esquerda na Figura 5.5. Então  $T_{U_2}$  é criada a partir de  $T_H$  colocando a folha  $l_1$  um nível abaixo e, consequentemente, tornando sua paridade par. Então:

$$\begin{aligned}
 c(T_{U_2}) &= 5f_1 + 4f_2 + 3f_3 + 2f_4 + f_5 \\
 &= c(T_H) + f_1 \\
 &= \left(1 + \frac{f_1}{c(T_H)}\right)c(T_H) \\
 &= \left(1 + \frac{f_1}{4f_1 + 4f_2 + 3f_3 + 2f_4 + f_5}\right)c(T_H) \\
 &\leq \left(1 + \frac{f_1}{14f_1}\right)c(T_H) = \frac{15}{14}c(T_H) \\
 &\leq \frac{7}{6}c(T_H).
 \end{aligned}$$

*Caso 2.* Agora seja  $T_H$  a árvore do centro na Figura 5.5.  $T_{U_2}$  é criada a partir de  $T_H$  colocando as folhas  $l_1$  e  $l_4$  um nível abaixo, tornando suas paridades pares, e depois trocando as folhas  $l_2$  e  $l_4$ . Portanto:

$$\begin{aligned}
 c(T_{U_2}) &= 4f_1 + 4f_2 + 3f_3 + 3f_4 + f_5 \\
 &= c(T_H) + f_1 + f_2 \\
 &= \left(1 + \frac{f_1 + f_2}{c(T_H)}\right)c(T_H) \\
 &= \left(1 + \frac{f_1 + f_2}{3f_1 + 3f_2 + 3f_3 + 3f_4 + f_5}\right)c(T_H) \\
 &\leq \left(1 + \frac{f_1 + f_2}{6(f_1 + f_2) + f_5}\right)c(T_H) \\
 &\leq \left(1 + \frac{f_1 + f_2}{6(f_1 + f_2)}\right)c(T_H) \\
 &\leq \frac{7}{6}c(T_H).
 \end{aligned}$$

*Caso 3.* Finalmente, seja  $T_H$  a árvore da direita na Figura 5.5.  $T_{U_2}$  é criada a partir de  $T_H$  movendo as folhas  $l_1$  e  $l_4$  para o nível inferior, tornando suas paridades pares, e intercambiando as folhas  $l_4$  e  $l_3$ . Portanto:

$$\begin{aligned}
 c(T_{U_2}) &= 4f_1 + 3f_2 + 3f_3 + 2f_4 + 2f_5 \\
 &= c(T_H) + f_1 + f_3 \\
 &= \left(1 + \frac{f_1 + f_3}{c(T_H)}\right)c(T_H) \\
 &= \left(1 + \frac{f_1 + f_3}{3f_1 + 3f_2 + 2f_3 + 2f_4 + 2f_5}\right)c(T_H) \\
 &\leq \left(1 + \frac{f_1 + f_3}{6(f_1 + f_3)}\right)c(T_H) \\
 &\leq \frac{7}{6}c(T_H).
 \end{aligned}$$

Suponhamos agora  $n > 5$ . Notemos que a folha  $l_n$  é a mais próxima da raiz. Consideremos três possibilidades para sua profundidade,  $d_n$ :  $d_n = 1$ ,  $d_n = 2$  e  $d_n > 2$ . Vamos mostrar, em cada caso, que a hipótese é válida.

*Caso a.*  $d_n = 1$ .

Neste caso,  $T_H$  é a árvore ilustrada na Figura 5.6. A subárvore  $T_{H_{21}}$  tem  $n_1 \geq 5$  folhas de codificação.  $T_{U_2}$  tem forma análoga a  $T_H$ . Seja  $T_{U_{21}}$  a subárvore de  $T_{U_2}$  correspondente a  $T_{H_{21}}$ . O resultado desejado pode ser provado por indução.

Dado que o número de folhas de codificação de  $T_{U_{21}}$  é maior que 4, suponhamos que  $c(T_{U_{21}}) \leq \frac{7}{6}c(T_{H_{21}})$ . Como  $c(T_{U_2}) = 1 + c(T_{U_{21}})$ , segue-se que:

$$\begin{aligned} c(T_{U_2}) &\leq 1 + \frac{7}{6}c(T_{H_{21}}) \\ &\leq \frac{7}{6}(1 + c(T_{H_{21}})) \\ &= \frac{7}{6}c(T_H). \end{aligned}$$

Consequentemente, o resultado continua válido para  $T_H$  e  $T_{U_2}$ .

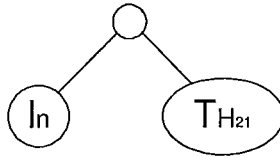


Figura 5.6: Árvore de Huffman para  $n > 5$ ,  $d_n = 1$

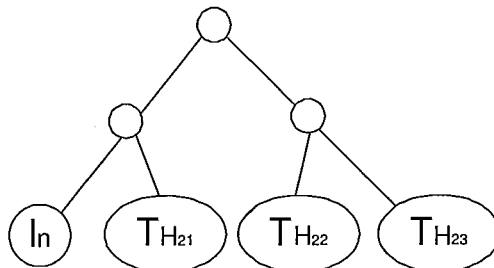


Figura 5.7: Árvore de Huffman para  $n > 5$ ,  $d_n = 2$

*Caso b.*  $d_n = 2$ .

Neste caso,  $T_H$  pode ser considerada como a árvore ilustrada na Figura 5.7, onde  $l_n$  é uma folha e  $T_{H_{21}}$ ,  $T_{H_{22}}$  e  $T_{H_{23}}$  são subárvores disjuntas contendo  $n_1$ ,  $n_2$

e  $n_3$  folhas, respectivamente. Sejam  $x, y$  and  $z$  as probabilidades acumuladas das folhas das subárvores  $T_{H_{21}}$ ,  $T_{H_{22}}$  e  $T_{H_{23}}$ , respectivamente. Claramente,  $f_n + x + y + z = 1$ . Devemos considerar 2 subcasos:

*Subcaso b.1.*  $T_{H_{21}}$  não é folha,  $T_{H_{22}}$  é folha ímpar e  $T_{H_{23}}$  é folha par.

Neste caso,  $T_{U_2}$  é obtida de  $T_H$  movendo  $T_{H_{22}}$  um nível abaixo e modificando  $T_{H_{21}}$  com as mesmas regras da Heurística 2, para obter  $T_{U_{21}}$ . Então:

$$\begin{aligned} c(T_{U_2}) &= 2 * (f_n + x + y + z) + c(T_{U_{21}}) + y \\ &= 2 + c(T_{U_{21}}) + y. \end{aligned}$$

Pelo Lema 7.3.1,  $c(T_{U_{21}}) \leq c(T_{H_{21}}) + x/2$ . Como  $c(T_H) = 2 + c(T_{H_{21}})$ , segue-se que:

$$\begin{aligned} c(T_{U_2}) &\leq 2 + c(T_{H_{21}}) + x/2 + y \\ &= (1 + \frac{x/2+y}{2+c(T_{H_{21}})})c(T_H). \end{aligned}$$

Dado que  $c(T_{H_{21}}) \geq x$ , temos  $c(T_{U_2}) \leq (1 + \frac{x/2+y}{2+x})c(T_H) = (\frac{3}{2} + \frac{y-1}{2+x})c(T_H)$ . A fração  $\frac{y-1}{x+2}$  tem seu valor máximo quando  $y = z = f_n$ . Isto implica  $x = 1 - 3y$ , e, conseqüentemente:

$$\begin{aligned} c(T_{U_2}) &\leq (\frac{3}{2} + \frac{y-1}{1-3y+2})c(T_H) \\ &= (\frac{3}{2} - \frac{1}{3})c(T_H) \\ &= \frac{7}{6}c(T_H). \end{aligned}$$

*Subcaso b.2.* Neste caso,  $T_{H_{21}}$  e  $T_{H_{22}}$  não são folhas.  $T_{H_{23}}$  pode ser folha ou uma subárvore com mais de uma folha.

$T_{U_2}$  é obtida a partir de  $T_H$  aplicando a Heurística 2 às subárvores  $T_{H_{21}}$ ,  $T_{H_{22}}$  e  $T_{H_{23}}$  para obter  $T_{U_{21}}$ ,  $T_{U_{22}}$  e  $T_{U_{23}}$ , respectivamente. Considerando que a folha  $l_n$  não é modificada pela heurística, temos, pelo Lema 7.3.1:

$$\begin{aligned} c(T_{U_2}) &\leq c(T_H) + \sum_{i=1}^{n-1} f_i/2 \\ &= c(T_H) + (x + y + z)/2 \\ &= (1 + \frac{x+y+z}{2c(T_H)})c(T_H) \\ &= (1 + \frac{x+y+z}{2(2+f_n+c(T_{H_{21}})+c(T_{H_{22}})+c(T_{H_{23}}))})c(T_H) \end{aligned}$$

Dado que  $c(T_{H_{21}}) + c(T_{H_{22}}) + c(T_{H_{23}}) \geq x + y + z < 1$ , temos:



$$\begin{aligned}
c(T_{U_2}) &\leq \left(1 + \frac{x+y+z}{2(2+f_n+x+y+z)}\right)c(T_H) \\
&= \left(1 + \frac{1}{2(2+1)}\right)c(T_H) \\
&= \left(1 + \frac{1}{6}\right)c(T_H) \\
&= \frac{7}{6}c(T_H).
\end{aligned}$$

Caso c.  $d_n > 2$ .

Neste caso, temos  $c(T_H) \geq 3$  e  $c(T_{U_2}) \leq c(T_H) + 1/2$ . Consequentemente:

$$\begin{aligned}
c(T_{U_2}) &\leq \left(1 + \frac{1}{2c(T_H)}\right)c(T_H) \\
&\leq \left(1 + \frac{1}{6}\right)c(T_H) \\
&= \frac{7}{6}c(T_H).
\end{aligned}$$

Em todos os casos,  $c(T_{U_2}) \leq \frac{7}{6}c(T_H)$ . ■

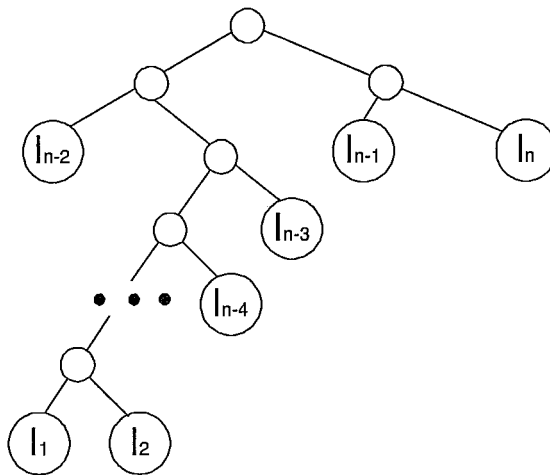


Figura 5.8: Família especial de árvores de Huffman

O limite dado pelo Teorema 5.3.2 não pode ser melhorado. Mostraremos isso pela exibição de uma família infinita  $\mathcal{F}$  de árvores de Huffman onde, dada uma aproximação  $\epsilon > 0$ , sempre podemos achar uma árvore  $T_H \in \mathcal{F}$  tal que, para a correspondente árvore par  $T_{U_2}$  obtida pela Heurística 2, temos  $\frac{7}{6}c(T_H) - c(T_{U_2}) \leq \epsilon$ .

Esta família é ilustrada na Figura 5.8. Cada árvore da família é caracterizada por 2 parâmetros:  $n$  e  $q$ ,  $n \geq 4$  e  $\frac{2}{5} \geq q > 0$ . Cada árvore tem  $n$  folhas e

profundidade  $n - 2$ . Três folhas ( $l_{n-2}$ ,  $l_{n-1}$  e  $l_n$ ) têm profundidade 2 e probabilidade  $\frac{1-q}{3}$ . As restantes  $n - 3$  folhas  $l_1, \dots, l_{n-3}$  estão distribuídas na árvore da seguinte maneira: uma folha, ( $l_1$ ), tem profundidade  $n - 2$  e probabilidade  $\frac{q}{2^{n-4}}$ ;  $n - 4$  folhas ( $l_i$ ,  $2 \leq i \leq n - 3$ ) têm profundidades  $n - i$  e probabilidades  $\frac{q}{2^{n-i-2}}$ , respectivamente. Nesta família de árvores, há somente duas folhas de paridade ímpar:  $l_1$  e  $l_{n-1}$ .

**Teorema 5.3.3** *Seja  $T_H$  uma árvore pertencente a  $\mathcal{F}$ . Então  $T_H$  é uma árvore de Huffman.*

**Demonstração:** Seja  $A_j = \sum_{i=1}^j f_i$ . Notemos que  $A_{n-3}$  satisfaz a:

$$A_{n-3} = \frac{q}{2^{n-4}} + \sum_{i=2}^{n-3} \frac{q}{2^{n-i-2}} = \frac{q}{2^{n-4}} \left( 1 + \frac{2^{n-2} - 2^2}{4} \right) = q.$$

Portanto, a soma de probabilidades em  $T_H$  é  $3\frac{1-q}{3} + q = 1$ . Uma condição necessária para  $T_H$  ser uma árvore de Huffman é que  $A_{n-3}$  não pode ser maior que  $f_{n-1} + f_n$ . Consequentemente, devemos ter  $q \leq 2\frac{1-q}{3}$ , o que implica  $q \leq \frac{2}{5}$ .

Além disso, vamos mostrar que  $A_{i-1} = f_i$ , para  $1 < i \leq n - 3$ :

$$\begin{aligned} A_{i-1} &= \frac{q}{2^{n-4}} + \sum_{j=2}^{i-1} \frac{q}{2^{n-j-2}} = \frac{q}{2^{n-2}} \left( 4 + \sum_{j=2}^{i-1} 2^j \right) = \\ &= \frac{q}{2^{n-2}} (4 + 2^i - 4) = \frac{q}{2^{n-i-2}} = f_i. \end{aligned}$$

Consequentemente, a folha  $l_i$  é corretamente agregada à subárvore contendo as folhas  $l_1, \dots, l_{i-1}$  durante o processo de criação de  $T_H$ . Portanto,  $T_H$  é uma árvore de Huffman, de fato. ■

**Teorema 5.3.4** *Dado  $\epsilon > 0$ , existe  $T_H \in \mathcal{F}$  tal que a correspondente árvore par  $T_{U_2}$ , obtida a partir de  $T_H$  pela aplicação da Heurística 2, satisfaz a  $\frac{7}{6}c(T_H) - c(T_{U_2}) \leq \epsilon$ .*

**Demonstração:** Seja  $T_H \in \mathcal{F}$ . Então:

$$c(T_H) = (n - 2) \frac{q}{2^{n-4}} + \sum_{i=2}^{n-3} (n - i) \frac{q}{2^{n-i-2}} + 3 \cdot 2 \frac{1 - q}{3}$$

$$= \frac{2^{n-2}(1+q) - 4q}{2^{n-3}}.$$

Agora, observemos que as folhas  $l_2, \dots, l_{n-3}$  de  $T_H$  têm paridade par, e as folhas  $l_1$  e  $l_{n-1}$  têm paridade ímpar (ver Figura 5.8). Então, a única diferença da árvore par  $T_{U_2}$  obtida a partir de  $T_H$  pela aplicação da Heurística 2 é que as folhas  $l_1$  e  $l_{n-1}$  de  $T_{U_2}$  são colocadas um nível abaixo. Portanto:

$$c(T_{U_2}) = c(T_H) + \frac{q}{2^{n-4}} + \frac{1-q}{3} = c(T_H) + \frac{3q + 2^{n-4}(1-q)}{3 \cdot 2^{n-4}}.$$

Dado  $\epsilon > 0$ , queremos escolher  $q$  tal que  $\frac{7}{6}c(T_H) - c(T_{U_2}) \leq \epsilon$  seja satisfeita. Devemos ter, então:

$$\frac{2^{n-2}(1+q) - 4q}{6 \cdot 2^{n-3}} - \frac{3q + 2^{n-4}(1-q)}{3 \cdot 2^{n-4}} \leq \epsilon$$

que é equivalente a

$$\frac{q(2^{n-5} - 1)}{3 \cdot 2^{n-5}} \leq \epsilon$$

Dado que  $\frac{(2^{n-5}-1)}{2^{n-5}} < 1$ , basta escolher  $q \leq \min(\frac{2}{5}, 3\epsilon)$ . ■

Encontramos um limite superior exato para a diferença de custo entre uma árvore par obtida pela Heurística 2 e uma árvore de Huffman correspondente. Claramente, este também é um limite superior (possivelmente não exato) para a diferença de custo entre uma árvore par ótima e a correspondente árvore de Huffman. Entretanto, permanece aberta a determinação de um limite exato para esta diferença.

A seguir mostraremos uma série de resultados experimentais.

## 5.4 Resultados experimentais

Alguns resultados experimentais relacionados a custos estão sumarizados nas Tabelas 5.1 a 5.3. As tabelas apresentam os custos de árvores obtidas pelos diversos algoritmos, para vários valores de  $n$ , número de símbolos.

A simbologia usada é a seguinte:  $c(T_H)$  refere-se ao custo da árvore de Huffman;  $c(T_E)$  é o custo da árvore par ótima;  $c(T_{U_1})$  é o custo da árvore obtida pela Heurística 1 e  $c(T_{U_2})$  é o custo da árvore obtida pela Heurística 2.

As tabelas foram obtidas através de programas escritos em Pascal, rodados em um computador Pentium IV com 1.8 GHz e 256M RAM.

$n$	$c(T_H)$ (A)	$c(T_E)$ (B)	$c(T_{U_1})$ (C)	$c(T_{U_2})$ (D)	$\frac{B-A}{A}$ %	$\frac{C-A}{A}$ %	$\frac{D-A}{A}$ %
64	6.00	6.50	6.50	6.50	8.3	8.3	8.3
128	7.00	7.50	7.50	7.50	7.1	7.1	7.1
192	7.67	8.00	8.17	8.00	4.4	6.5	4.4
256	8.00	8.50	8.50	8.50	6.3	6.3	6.3
320	8.40	8.80	8.90	8.80	4.8	6.0	4.8
384	8.67	9.00	9.17	9.00	3.9	5.8	3.9
448	8.86	9.29	9.36	9.29	4.8	5.7	4.8
512	9.00	9.50	9.50	9.50	5.6	5.6	5.6
576	9.22	9.67	9.72	9.67	4.8	5.4	4.8
640	9.40	9.80	9.90	9.80	4.3	5.3	4.3
704	9.55	9.91	10.05	9.91	3.8	5.2	3.8
768	9.67	10.00	10.17	10.00	3.5	5.2	3.5
832	9.77	10.15	10.27	10.15	3.9	5.1	3.9
896	9.86	10.29	10.36	10.29	4.4	5.1	4.4
960	9.93	10.40	10.43	10.40	4.7	5.0	4.7
1024	10.00	10.50	10.50	10.50	5.0	5.0	5.0

Tabela 5.1: Comparações de custo para probabilidades uniformes

Nas Tabelas 5.1 e 5.2 comparamos os custos  $c(T_H)$ ,  $c(T_E)$ ,  $c(T_{U_1})$  e  $c(T_{U_2})$ , para  $n$  na faixa de 64 a 1024. Na Tabela 5.1 usamos probabilidades uniformes, e na Tabela 5.2, probabilidades arbitrárias, obtidas através da rotina padrão do Pascal de geração de números pseudoaleatórios na faixa 1 a 10000 (não encontramos variações significativas mudando esta faixa). Todas as probabilidades foram normalizadas, tal que a soma total fosse 1. Na Tabela 5.3 comparamos as duas heurísticas com Huffman, para valores grandes de  $n$ , na faixa 1000 a

$n$	$c(T_H)$ (A)	$c(T_E)$ (B)	$c(T_{U_1})$ (C)	$c(T_{U_2})$ (D)	$\frac{B-A}{A}$ %	$\frac{C-A}{A}$ %	$\frac{D-A}{A}$ %
64	5.82	5.97	6.38	6.03	2.7	9.7	3.7
128	6.76	6.88	7.28	6.92	1.9	7.8	2.4
192	7.33	7.52	7.84	7.59	2.7	7.0	3.6
256	7.75	7.87	8.25	7.90	1.6	6.5	2.0
320	8.08	8.20	8.57	8.30	1.5	6.1	2.7
384	8.34	8.54	8.85	8.60	2.5	6.1	3.1
448	8.52	8.68	9.04	8.73	1.9	6.0	2.4
512	8.72	8.84	9.22	8.87	1.3	5.8	1.7
576	8.92	9.03	9.39	9.09	1.2	5.3	2.0
640	9.10	9.22	9.60	9.31	1.4	5.6	2.3
704	9.20	9.33	9.69	9.41	1.4	5.4	2.4
768	9.33	9.53	9.83	9.58	2.1	5.3	2.7
832	9.44	9.63	9.93	9.67	2.0	5.2	2.4
896	9.57	9.73	10.08	9.77	1.7	5.3	2.1
960	9.64	9.78	10.14	9.82	1.5	5.2	1.9
1024	9.75	9.87	10.24	9.89	1.3	5.1	1.5

Tabela 5.2: Comparações de custo para probabilidades arbitrárias

100000.

O principal resultado observado na Tabela 5.1 é que, para probabilidades uniformes, a Heurística 2 iguala-se ao algoritmo exato, ao passo que a Heurística 1, não. Esse resultado será provado no final do capítulo.

Pode também ser observada a pequena diferença de custos entre o algoritmo de Huffman e os demais métodos e também o decréscimo relativo dessa diferença, à medida que  $n$  cresce.

Podemos ainda confirmar os resultados teóricos discutidos na seção 4.2.3 sobre as diferenças de custo entre árvores pares ótimas e árvores de Huffman. Elas variam no intervalo  $[1/3, 1/2]$ , sendo máximas  $(1/2)$  quando o número de símbolos é  $n = 2^k$  para algum inteiro  $k$ , e mínimas  $(1/3)$ , quando  $n = 3 \cdot 2^k$ .

Agora, examinemos os resultados apresentados na Tabela 5.2, para probabilidades arbitrárias. Primeiro, comparemos dados das tabelas 5.1 e 5.2. Podemos ver que todos os dados nas colunas 2 a 5 da Tabela 5.2 são inferiores aos correspondentes da Tabela 5.1; isto é um resultado teórico esperado, dado que os custos têm a tendência a decrescer à medida que as probabilidades se desbalanceiam.

A diferença relativa entre  $c(T_E)$  e  $c(T_H)$  decresce consideravelmente, quando  $n$  cresce. O mesmo comportamento se dá para a Heurística 2, o que sugere que ela também é bem aplicada a essa situação, mesmo não se igualando à solução ótima.

Entretanto, para a Heurística 1, o comportamento é diferente. Tanto em termos de valores absolutos quanto relativos, as diferenças crescem. Portanto, a Heurística 2 é muito melhor que a 1, nesta situação.

A Tabela 5.3 ilustra os custos obtidos para valores grandes de  $n$  e probabilidades arbitrárias. Comparam-se os custos  $c(T_H)$  com  $c(T_{U_1})$  e  $c(T_{U_2})$ .

Os principais resultados obtidos pela Tabela 5.2 são confirmados, ou seja, a Heurística 2 é muito melhor que a 1.

Além disso, as diferenças de custo entre as duas heurísticas e Huffman continuam a decrescer, tornando-se desprezíveis, para  $n$  muito grande.

Considerando as três tabelas, podemos observar que há uma diferença entre o limite superior teórico apresentado na seção 5.3 e os resultados experimentais. As diferenças de custo encontradas são, no máximo, 5%, para  $n$  grande, situando-se bem abaixo do limite teórico que é de 16.7%.

Finalmente, vamos formalizar o resultado observado de que a Heurística 2 obtém uma árvore ótima, para probabilidades uniformes.

**Teorema 5.4.1** *A árvore par  $T_U$ , obtida pela Heurística 2, é ótima, para probabilidades uniformes.*

**Demonstração:** Seja  $T_H$  a árvore de Huffman,  $T_E$  a árvore par ótima,  $T_U$  a árvore obtida pela Heurística 2 e  $P(T_H), P(T_E), P(T_U)$  os caminhos de codi-

ficação respectivos, para  $n > 2$  símbolos. Na situação de probabilidades uniformes, as folhas de  $T_H$  sempre estão situadas nos dois últimos níveis dessa árvore. A única melhoria efetiva, durante a aplicação da Heurística 2 é a Melhoria 2, ou seja, a troca, quando possível, de folhas ímpares do penúltimo nível com subárvores pares desse mesmo nível. Consideremos 3 casos.

*Caso 1.*  $n$  é da forma  $n = 2^k$ ,  $k$  inteiro. Neste caso, temos  $\lceil \log n \rceil = k$  e  $\lceil \log \frac{n}{3} \rceil = k - 1$ . Portanto,

$$P(T_H) = n(k + 1) - 2^k.$$

$$P(T_E) = n(k - 1 + 3) - 3 \cdot 2^{k-1} = n(k + 2) - 3 \cdot 2^{k-1}.$$

$T_U$  é obtida a partir de  $T_H$  ampliando em 1 o nível das folhas ímpares, que são a metade das folhas de  $T_H$ . Então,

$$P(T_U) = P(T_H) + n/2 = n(k + 2) - 2^k - n/2 = n(k + 2) - 2^k - 2^{k-1} = n(k + 2) - 3 \cdot 2^{k-1} = P(T_E).$$

Nos dois casos a seguir, chamaremos  $lp$  e  $lu$  o número de folhas de  $T_H$  existentes no penúltimo e último nível da árvore, respectivamente. A profundidade de  $T_H$  é dada por  $d = \lceil \log n \rceil$ . Temos as relações:  $lp + lu = n$  e  $lp + lu/2 = 2^{d-1}$  o que leva a:  $lp = 2^d - n$ ,  $lu = 2n - 2^d$ .

*Caso 2.*  $n$  é da forma  $n = 2^k + a$ ,  $k, a$  inteiros,  $2^{k-1} < a < 2^k$ . Neste caso,  $d = k + 1$ ,  $\lceil \log \frac{n}{3} \rceil = k$ ,  $lp = 2^{k+1} - n$  e  $lu = 2n - 2^{k+1}$ . O número de nós internos do penúltimo nível é dado por  $lip = 2^k - lp = 2^k - 2^{k+1} + 2^k + a = a > 2^{k-1}$ . Ou seja,  $lip$  é mais da metade dos nós do penúltimo nível. Conseqüentemente, a aplicação da Melhoria 2 transforma todas as folhas desse nível em folhas pares. O número de folhas ímpares restantes é metade daquelas do último nível, ou seja,  $lu/2 = n - 2^k$ . Temos, então:

$$P(T_H) = n(k + 1 + 1) - 2^{k+1} = n(k + 2) - 2^{k+1}.$$

$$P(T_E) = n(k + 3) - 3 \cdot 2^k.$$

$$P(T_U) = P(T_H) + n - 2^k = n(k + 2) - 2^{k+1} + n - 2^k = n(k + 3) - 3 \cdot 2^k = P(T_E).$$

*Caso 3.*  $n$  é da forma  $n = 2^k + a$ ,  $k, a$  inteiros,  $1 < a \leq 2^{k-1}$ . Neste caso,  $d = k + 1$ ,  $\lceil \log \frac{n}{3} \rceil = k - 1$ ,  $lp = 2^{k+1} - n$  e  $lu = 2n - 2^{k+1}$ . O número de

folhas do penúltimo nível é não inferior ao de nós internos pois  $lip = a \leq 2^{k-1}$ . Então, o número de folhas ímpares restantes nesse nível, após a aplicação da Melhoria 2, é  $2^{k-1} - lip = 2^{k-1} - (n - 2^k) = 3 \cdot 2^{k-1} - n$ . Também aqui, metade das folhas do último nível é de folhas ímpares. Esse número é dado por  $lu/2 = n - 2^k$ . Temos:

$$P(T_H) = n(k + 1 + 1) - 2^{k+1} = n(k + 2) - 2^{k+1}.$$

$$P(T_E) = n(k - 1 + 3) - 3 \cdot 2^{k-1} = n(k + 2) - 3 \cdot 2^{k-1}.$$

$$P(T_U) = P(T_H) + 3 \cdot 2^{k-1} - n + n - 2^k = n(k + 2) - 2^{k+1} + 2^{k-1} = n(k + 2) - 3 \cdot 2^{k-1} = P(T_E).$$

Em todos os casos,  $T_U$  tem mesmo caminho de codificação que  $T_E$  e, portanto, é ótima.

■



$n$	$c(T_H)$ (A)	$c(T_{U_1})$ (B)	$c(T_{U_2})$ (C)	$\frac{B-A}{A}$ %	$\frac{C-A}{A}$ %
1000	9.71	10.22	9.87	5.2	1.6
2000	10.72	11.21	10.87	4.6	1.5
3000	11.31	11.81	11.57	4.5	2.3
4000	11.72	12.22	11.87	4.3	1.3
5000	12.05	12.55	12.26	4.1	1.8
10000	13.04	13.55	13.25	3.9	1.6
15000	13.61	14.11	13.80	3.7	1.4
20000	14.05	14.55	14.25	3.6	1.5
25000	14.36	14.86	14.61	3.5	1.8
30000	14.62	15.12	14.81	3.4	1.3
35000	14.85	15.35	15.01	3.4	1.1
40000	15.05	15.55	15.25	3.3	1.4
45000	15.21	15.71	15.45	3.3	1.6
50000	15.36	15.86	15.61	3.3	1.7
55000	15.49	15.99	15.71	3.2	1.4
60000	15.62	16.12	15.80	3.2	1.2
65000	15.74	16.24	15.89	3.2	1.0
70000	15.85	16.35	16.01	3.2	1.0
75000	15.95	16.45	16.14	3.1	1.2
80000	16.04	16.54	16.25	3.1	1.3
85000	16.13	16.63	16.35	3.1	1.4
90000	16.21	16.71	16.45	3.1	1.5
95000	16.29	16.79	16.54	3.1	1.5
100000	16.36	16.86	16.61	3.1	1.6

Tabela 5.3: Comparações de custo para probabilidades arbitrárias,  $n$  grande

# Capítulo 6

## Detecção de Erros em Árvores

### Pares

Nesta seção desenvolvemos um modelo probabilístico para avaliar a capacidade de detecção de erros de uma árvore par. Apresentamos, também, resultados experimentais para validar o modelo. No final do capítulo discutimos a compactação de linguagem natural utilizando árvores pares.

#### 6.1 Um modelo probabilístico para detecção de erros em árvores pares

Suponhamos que tenha ocorrido um ou mais erros em uma mensagem codificada, isto é, um ou mais bits foram trocados.

Após o primeiro erro, podemos ter uma sequência de símbolos decodificados erroneamente. Por exemplo, seja um código par simples com três símbolos  $C = \{a, b, c\}$ , com codificações respectivas  $\{0, 11 \text{ e } 101\}$ . A mensagem 'ababccb' é codificada como '01101110110111'. Se for introduzido um erro no quarto bit, a mensagem passa para '01111110110111'. O processo de decodificação interpretaria os 13 primeiros bits como 'abbbabab' e no final seria indicado um erro, pois não há símbolos correspondendo ao último bit '1' dessa mensagem espúria. Entretanto, o erro foi propagado a partir do terceiro símbolo e só foi

descoberto após a decodificação do oitavo símbolo.

O modelo descrito a seguir avalia a probabilidade de uma sequência de  $k$  símbolos serem decodificados erroneamente, utilizando uma árvore par. Estima-se, também, a probabilidade de detecção de erros ocorridos.

Seja  $s_j$  o  $j$ -ésimo símbolo em uma mensagem codificada, aquele onde o primeiro erro foi introduzido, e  $T$ , a árvore par correspondente usada para decodificar a mensagem.

Inicialmente calcularemos a probabilidade  $PP_j(T, k)$  de que os próximos  $k$  símbolos (incluindo  $s_j$ ) sejam erroneamente decodificados.  $PP_j(T, k)$  é a *probabilidade de propagação de erros* pelos próximos  $k - 1$  símbolos.

Após a introdução do primeiro erro, a sequência de bits restante pode ser considerada como um “string” aleatório. Em termos da árvore de decodificação, isto significa que cada desvio na árvore tem a mesma probabilidade, que é  $1/2$ .

Consideremos que o primeiro bit errado corresponda à raiz da árvore. Então, a probabilidade da folha  $l_j$  ser atingida é  $2^{-d_j}$ , onde  $d_j$  é a profundidade de  $l_j$ .

Definimos dois parâmetros para  $T$ :  $Pt_c(T)$  e  $Pt_e(T)$ . O parâmetro  $Pt_c(T)$  é a probabilidade de alguma folha de codificação ser atingida a partir da raiz, e seu valor é  $Pt_c(T) = \sum_{j=1}^n 2^{-d_j}$ . Analogamente,  $Pt_e(T)$  é a probabilidade de alguma folha de erro ser atingida e é dada por  $Pt_e(T) = \sum_{j=1}^{Ne(n)} 2^{-d_j}$ , onde  $Ne(n)$  é o número de folhas de erro de  $T$  e  $d_j$  é a profundidade da  $j$ -ésima folha de erro. Claramente,  $Pt_c(T) + Pt_e(T) = 1$ .

A probabilidade de propagação do erro ocorrido no  $i$ -ésimo símbolo da mensagem, pelos próximos  $k$  símbolos(inclusive) é, então,  $PP_i(T, k) = Pt_c(T)^k = (1 - Pt_e(T))^k$ , decrescendo exponencialmente com  $k$ . Note-se que considerar que o primeiro erro ocorreu exatamente no bit correspondente à raiz somente aumenta esta probabilidade, uma vez que as folhas de erro estão situadas nos dois últimos níveis da árvore. Por isso, vamos desprezar o fato de que o erro pode ocorrer em outro ponto que não corresponda à raiz.

Vamos ver agora como estimar a capacidade de detecção de erros de  $T$ . Seja  $b$  o número médio de símbolos em uma mensagem. A probabilidade de detecção de erros de  $T$  para mensagens com  $b$  símbolos,  $P_m(T, b)$ , é o complemento da probabilidade de que os erros se propaguem até o fim da mensagem. Se os erros forem independentes e um erro ocorrer com probabilidade  $q$  pequena, então a probabilidade de ocorrência de erros é dada por  $\sum_{i=1}^B q^i \text{Comb}(B, i) \simeq q$ , onde  $B$  é o número de bits da mensagem. Como vemos, a probabilidade de ocorrer 1 erro domina a soma. Na sequência, consideramos apenas a ocorrência do primeiro erro. Seja:

$p_1(i, j)$  = probabilidade de que o  $j$ -ésimo símbolo da mensagem seja o símbolo  $s_i$ ;

$p_2(i)$  = probabilidade condicional de que o símbolo  $s_i$  seja o símbolo que contenha o primeiro bit errado, em caso de erro.

Como cada  $j$ -ésimo símbolo da mensagem pode ser um dos símbolos  $s_1$  a  $s_n$ , então o valor esperado para  $P_m(T, b)$  será aproximado por:

$$P_m(T, b) = 1 - \sum_{j=1}^b \sum_{i=1}^n p_1(i, j) \cdot p_2(i) \cdot PP_j(T, b + 1 - j)$$

Mas temos:  $p_1(i, j) = f_i$  e  $p_2(i) = u \cdot d_i$ , isto é, a probabilidade de que o símbolo  $s_i$  contenha o bit errado é proporcional ao tamanho de  $s_i$ . Então  $\sum_{i=1}^n p_1(i, k) \cdot p_2(i) = u \cdot \sum_{i=1}^n f_i d_i = v$ . Como estamos supondo que houve erro, a soma  $\sum_{j=1}^b \sum_{i=1}^n p_1(i, j) \cdot p_2(i) = b \cdot v$  deve ser 1. Portanto,  $v = 1/b$ . Com isso, a expressão fica simplificada para:

$$\begin{aligned} P_m(T, b) &= 1 - \sum_{j=1}^b PP_j(T, b + 1 - j) / b \\ &= 1 - (1 - Pt_e(T) - (1 - Pt_e(T))^{b+1}) / (b \cdot Pt_e(T)). \end{aligned}$$

Podemos refinar este modelo, considerando todos os erros introduzidos na mensagem e não apenas o primeiro. Como  $T$  detecta todos os erros em um número ímpar de bits, e supondo que pelo menos metade de todas as situações de erro sejam relacionadas a um número ímpar de erros, então podemos dividir a probabilidade de propagação de erros por 2.

Mas podemos ainda considerar que, no procedimento de decodificação do último símbolo, o processo pode não terminar em uma folha. Pode terminar

em um nó interno e, neste caso, o erro pode ser apontado. Como o número de nós internos é não inferior ao de folhas de codificação, dividiremos novamente a probabilidade de propagação de erros por 2.

Com esses refinamentos, temos:

$$\begin{aligned} P_m(T, b) &\geq 1 - \sum_{k=1}^b (1 - Pt_e(T))^{k-1} / 4b \\ &= 1 - (1 - (1 - Pt_e(T))^b) / (4b \cdot Pt_e(T)). \end{aligned}$$

A expressão acima mostra que, para estimar a capacidade de detecção de erros de uma árvore par  $T$ , o principal parâmetro a considerar é  $Pt_e(T)$ , que só depende das folhas de erro. Na sequência, exploramos os limites deste parâmetro, dados pelos dois próximos teoremas.

**Teorema 6.1.1** *Se  $T$  é uma árvore par ótima para  $n$  símbolos, com probabilidades uniformes, então  $1/16 \leq Pt_e(T) \leq 1/4$ .*

**Demonstração:** Por definição,  $Pt_e(T) = \sum_{i=1}^{Ne(T)} 2^{-d_i}$ ,

onde  $Ne(T)$  é o número de folhas de erro em  $T$  e  $d_i$  é a profundidade da folha de erro  $l_i$ .

Vimos, pelo corolário do Teorema 4.2.4, que, quando  $n > 5$ ,  $T$  é composta pelos 3 tipos de árvores mostradas na Figura 4.2. Em qualquer caso, para cada folha de erro, temos pelo menos duas folhas de codificação associadas: sua irmã e a irmã do pai. Então, considerando essas folhas de codificação, temos:

$$Pt_c(T) \geq \sum_{i=1}^{Ne(T)} (2^{-(d_i-1)} + 2^{-d_i}) = 3 \sum_{i=1}^{Ne(T)} 2^{-d_i} = 3Pt_e(T).$$

Como  $Pt_e(T) + Pt_c(T) = 1$ , temos  $Pt_e(T) + 3Pt_e(T) \leq 1$ . Portanto,

$$Pt_e(T) \leq 1/4.$$

Quanto ao limite inferior,  $1/16$ , ele ocorre quando  $T$  é composta apenas pelo terceiro tipo de árvore básica da mesma Figura 4.2, que é a que possui a menor relação  $Ne(T)/n$  onde  $Ne(T)$  é o número de folhas de erro de  $T$ . A árvore par formada apenas com essa estrutura básica tem  $Pt_e(T)$  mínimo.

Ela ocorre quando  $n = 2^k$ ,  $k > 1$  inteiro. Nesta situação, a árvore tem  $2^{k-2}$  folhas de erro, todas com profundidade máxima igual a  $k + 2$  (ver Capítulo 4). Portanto,

$$Pt_e(T) = 2^{k-2}2^{-(k+2)} = 2^{k-2-k-2} = 1/16.$$

Considerando os dois limites,  $1/16 \leq Pt_e(T) \leq 1/4$ . ■

**Teorema 6.1.2** *Se  $T$  é uma árvore par ótima para  $n$  símbolos, com probabilidades arbitrárias, então  $2^{-n} \leq Pt_e(T) \leq 1/4$ .*

**Demonstração:** O limite superior  $1/4$  já foi obtido no teorema anterior. Com respeito ao limite inferior, notar que a árvore par ótima  $T$  com mínimo  $Pt_e(T)$  é uma árvore especial, com profundidade igual a  $n$ . Em cada profundidade  $i \in \{1, \dots, n\}$ , há apenas uma folha de codificação (com paridade par). Consequentemente, haverá apenas uma folha de erro, no último nível. Portanto,  $Pt_e(T) = 2^{-n}$ .

Concluindo, para probabilidades arbitrárias, temos:

$$2^{-n} \leq Pt_e(T) \leq 1/4. \quad \blacksquare$$

Os limites dados pelos Teoremas 6.1.1 e 6.1.2 são para árvores pares ótimas. Os resultados para árvores pares obtidas pela Heurística 2, são dados pelo corolário a seguir.

**Corolário 6.1.3** *Se  $T$  é uma árvore par obtida pela Heurística 2, para  $n$  símbolos, então*

(1)  $1/8 \leq Pt_e(T) \leq 1/4$ , se as probabilidades são uniformes

(2)  $2^{-n} \leq Pt_e(T) \leq 1/4$ , se as probabilidades são arbitrárias

**Demonstração:** A única diferença que temos, em relação à situação dos teoremas 6.1.1 e 6.1.2 é no limite inferior, na situação de probabilidades uniformes. A Heurística 2 sempre usa subárvores isomorfas à primeira e à segunda

árvores da Figura 4.2. Se  $T$  é uma árvore par obtida pela Heurística 2, tendo  $Pt_e(T)$  mínimo, então  $T$  deve ser composta apenas por subárvores isomorfas à árvore com 3 folhas de codificação. Esta situação ocorre para  $n$  da forma  $n = 3 \cdot 2^k$ ,  $k \geq 0$ . Portanto,  $T$  tem  $2^k$  folhas de erro, todas com profundidade  $k + 3$ , como foi visto no Capítulo 4. Portanto, neste caso,

$$Pt_e(T) = 2^k \cdot 2^{-(k+3)} = 2^{k-k-3} = 1/8.$$

Considerando os dois limites,  $1/8 \leq Pt_e(T) \leq 1/4$ . ■

## 6.2 Resultados experimentais na detecção de erros

Nesta seção apresentamos alguns resultados experimentais sobre detecção de erros. A Tabela 6.1 ilustra valores teóricos e experimentais para  $P_m(T, b)$ , a probabilidade de detecção de erros de uma árvore par, considerando o modelo da seção anterior.

Na primeira coluna apresentamos uma faixa de valores para  $b$ , o número de símbolos em uma mensagem, variando de 10 a 5000. Na segunda coluna, são mostrados os valores teóricos para  $P_m(T, b)$ , de acordo com o modelo. Foi usado como base o valor  $Pt_e(T) = 0.1248$ , que é aproximadamente o valor médio para este parâmetro. A terceira coluna contém valores experimentais para  $P_m(T, b)$ , a probabilidade de detecção de erros, obtidos por simulação a partir de uma árvore par  $T$ , tendo  $Pt_e(T) = 0.1248$ . A simulação envolveu a geração de cerca de 20 milhões de erros aleatórios consistindo de um a vinte bits trocados, numa mensagem aleatória gerada com  $b$  símbolos.

Podemos observar o rápido crescimento da probabilidade de detecção de erros, à medida que o tamanho da mensagem aumenta. A probabilidade de detecção de erros é bastante alta para  $b \geq 100$ .

Pode também ser verificado que os valores experimentais de probabilidades de detecção de erros são maiores que aqueles previstos pelo modelo, especial-

$b$	$P_m(T, b)$	
	Teórico $(1 - (1 - 0.8752^b)/(0.4992 * b))$	Experimental
10	0.852500	0.903540
25	0.922732	0.987620
50	0.959987	0.990953
100	0.979968	0.994894
250	0.991987	0.998482
500	0.995994	0.999150
1000	0.997997	0.999595
2500	0.999199	0.999831
5000	0.999599	0.999922

Tabela 6.1: Probabilidades de detecção de erros

mente para valores baixos de  $b$ . Isto indica que o modelo utilizado pode ser aperfeiçoado. Mas as diferenças entre teoria e prática são bastante baixas para  $b > 100$ .

Conclui-se que, em média, árvores pares detectam muito bem erros para mensagens com pelo menos 100 símbolos. Entretanto, na próxima seção, mostraremos um problema existente na compressão de linguagem natural.

### 6.3 Compressão de linguagem natural usando árvores pares

Nesta seção, descrevemos um problema que aparece quando fazemos a compressão de linguagem natural, usando árvores pares.

Zipf [49] observou que a  $n$ -ésima palavra mais comum em uma linguagem natural parece ocorrer com uma frequência inversamente proporcional a  $n$ . Esta propriedade é conhecida como *Lei de Zipf*. Na distribuição de Zipf, dados  $n$  símbolos, a probabilidade de ocorrência do símbolo  $s_i$  é  $p_i = 1/(i.H_n)$ , onde  $H_n$  é o  $n$ -ésimo Número Harmônico. Vários autores usam esta lei como um



modelo teórico para as probabilidades de ocorrências das palavras em textos escritos em linguagem natural [18, 47]. Mais adiante nesta seção, mostraremos dados experimentais consistentes com esse modelo.

Caracterizaremos, parcialmente, as árvores pares para a distribuição de Zipf. Gutman [12] estudou as árvores de Huffman para essa distribuição, na situação particular em que  $n$  é da forma  $n = 2^{2^k} - 1$ ,  $k$  inteiro. Ele demonstrou que a profundidade da árvore é  $k + 2^k - 1$  e que a distribuição de folhas na árvore é a seguinte: no nível de profundidade  $k + i$ ,  $0 \leq i < 2^k$ , existem  $2^i$  folhas.

A única diferença da árvore par  $T$ , gerada pela Heurística 2, em relação à árvore de Huffman  $T_H$  é no último nível, onde metade das folhas são colocadas num novo nível criado após o último de  $T_H$ . Ao mesmo tempo, são criadas neste nível igual número de folhas de erro, como irmãs das folhas de codificação. Isto é uma consequência do procedimento dessa heurística para obter a árvore  $T$ : ela parte da árvore de Huffman e faz modificações na árvore sempre que o número de folhas em dado nível seja maior que o de nós internos, ou que haja folhas com paridade ímpar. Em todos os níveis, exceto no último, o número de folhas é sempre inferior ao de nós internos. Neste caso, a única modificação é trocar folhas de paridade ímpar com nós de paridade par. O único nível onde há mais folhas que nós internos é no último nível. Então metade das folhas, as de paridade ímpar, são colocadas num novo nível inferior e o mesmo número de folhas de erro é criado, como irmãs dessas folhas. Em  $T$  temos  $2^{2^k-2}$  folhas de erro, todas com profundidade  $k + 2^k$ . Consequentemente,

$$Pt_e(T) = 2^{2^k-2}/2^{k+2^k} = 2^{-k-2} = 2^{-\lceil \log \log n \rceil - 2}.$$

Esse resultado mostra que o parâmetro  $Pt_e$  das árvores pares correspondentes são muito baixos (por exemplo para  $n = 256$ ,  $Pt_e(T) = 0.03125$ ), o que indica baixo poder de detecção de erros. Resultados experimentais mostram que, para qualquer valor de  $n$ , esses parâmetros são igualmente baixos, situando-se na faixa (0.01 - 0.04). Isso traz implicações que serão comentadas adiante.

A seguir, apresentamos alguns resultados experimentais. Na Tabela 6.2 mostramos dados da compactação de sete conjuntos de textos escritos em duas linguagens naturais: Português e Inglês. Os três primeiros conjuntos são livros famosos de autores brasileiros. Os quatro últimos, parte de um conjunto de arquivos usados como testes de referência para sistemas de compressão de dados, e livremente disponíveis na página web *www.corpus.canterbury.ac.nz*. Para cada conjunto, apresentamos o número de palavras diferentes no texto, o custo  $c(T)$  da árvore par correspondente  $T$  e seu parâmetro  $Pt_e(T)$ . Os símbolos básicos para compactação foram as palavras dos textos. Também apresentamos o custo  $c(T_Z)$ , bem como o parâmetro  $Pt_e(T_Z)$  para a árvore par  $T_Z$  construída usando a distribuição de Zipf de probabilidades, correspondente ao número de palavras do texto. Todas as árvores pares foram construídas usando a heurística descrita no capítulo anterior.

Podemos notar que, embora não tenhamos uma correspondência exata entre as duas árvores, elas são muito próximas uma da outra, tanto em termos do custo quanto do parâmetro  $Pt_e$ . A distribuição de probabilidades de Zipf pode ser, então, um bom começo para explicar a distribuição de palavras em uma linguagem natural. Também observamos o importante fato de que o parâmetro  $Pt_e$  é muito baixo para ambas árvores, muito longe do valor médio que é 0.125. Isto significa uma pobre capacidade de detecção de erros, como veremos a seguir.

Na Seção 6.1 vimos que a probabilidade de detecção de erros de uma árvore par  $T$ ,  $P_m(T, b)$ , na decodificação de uma mensagem, com um número médio de símbolos  $b$ , pode ser expressa, aproximadamente, por:  $P_m(T, b) \geq 1 - (1 - (1 - Pt_e(T))^b) / (4b \cdot Pt_e(T))$ .

Na Tabela 6.3 apresentamos valores teóricos e experimentais das probabilidades de detecção de erros por árvores pares. Na primeira coluna é dado o número médio de símbolos na mensagem, variando de 10 a 5000. As duas próximas colunas referem-se a uma árvore par,  $T_1$ , com  $Pt_e(T_1) = 0.125$ , que é o valor médio esperado para esse parâmetro. As duas últimas colunas referem-se a uma árvore par,  $T_2$ , com  $Pt_e(T_2) = 0.0156$ , que é um valor próximo à

Texto	número de palavras	$c(T)$	$Pt_e(T)$	$c(T_Z)$	$Pt_e(T_Z)$
5 livros de Machado Assis	16399	9.92	0.0077	10.48	0.0156
Os Sertões (E. Cunha)	21934	10.97	0.0219	10.70	0.0156
6 livros de José de Alencar	16475	10.22	0.0154	10.50	0.0156
Alice (Lewis Carrol)	2573	8.66	0.0085	8.85	0.0202
Paradise Lost (J. Milton)	9063	10.03	0.0173	10.01	0.0171
Project Guttenberg	5560	9.56	0.0091	9.59	0.0184
Three Muskeeters(A.Dumas)	10494	9.29	0.0039	10.21	0.0168

Tabela 6.2: Parâmetros para árvores pares usadas na compressão de arquivos de texto

$b$	$Pt_m(T_1, b)$ ( $Pt_e(T_1) = 0.125$ )		$Pt_m(T_2, b)$ ( $Pt_e(T_2) = 0.0156$ )	
	teórica	experimental	teórica	experimental
10	0.852615	0.903540	0.766839	0.791726
25	0.922840	0.987620	0.791653	0.793113
50	0.960050	0.990953	0.825512	0.826450
100	0.980000	0.994894	0.873008	0.877410
250	0.992000	0.998482	0.937156	0.935279
500	0.996000	0.999150	0.967961	0.979352
1000	0.998000	0.999595	0.983974	0.990782
2500	0.999200	0.999831	0.993590	0.997993
5000	0.999600	0.999922	0.996795	0.998746

Tabela 6.3: Comparação de probabilidades de detecção de erros por árvores pares

média dos diversos parâmetros de árvores correspondentes a compressão de linguagem natural, obtidos na Tabela 6.2.

Podemos observar que há uma relação muito próxima entre os valores teóricos e experimentais para ambas as árvores. Os resultados experimentais  $T_1$  são ligeiramente maiores do que os teóricos correspondentes, ao passo que, para  $T_2$ , este fato só ocorre para  $b \geq 500$ , e com comportamento oposto para valores baixos de  $b$ . Para  $T_1$ , temos probabilidades de detecção de erro aceitáveis ( $\geq 99.4\%$ ) para  $b \geq 100$ , ao passo que, para  $T_2$ , este fato só ocorre  $b > 1000$ .

Este resultado mostra que podemos ter problemas na compactação de linguagem natural usando árvores pares, quando precisamos de alta confiabilidade na detecção de erros, a menos que estejamos lidando com mensagens muito grandes. Desta forma, parece importante procurarmos alternativas para contornar o fato. Esta questão será considerada no próximo capítulo.

# Capítulo 7

## Árvores $\alpha$ -pares

Neste capítulo vamos definir as árvores  $\alpha$ -pares, uma extensão das árvores pares, que permitem uma variação no poder de detecção de erros, de acordo com um dado parâmetro  $\alpha$ . Vamos caracterizar as árvores para o caso especial em que  $\alpha = 1$  e também apresentar um algoritmo polinomial para obter árvores  $\alpha$ -pares ótimas.

A motivação para a definição dessas árvores vem do paradoxo existente quando se considera compactação versus poder de detecção de erros em árvores pares: quanto mais uma árvore par é próxima da árvore de Huffman, em termos de compactação, mais pobre é sua habilidade em detectar erros. Isso foi visto no capítulo anterior, quando temos a distribuição de Zipf nas probabilidades. Para melhorar o poder de detecção de erros, a saída é acrescentar mais redundância às árvores pares. Para fazer isso, vamos modificar a definição dessas árvores, forçando a existência de mais folhas de erro. Em especial, é importante a possibilidade de termos folhas de erro com qualquer profundidade, visto que, em árvores pares ótimas, folhas de erro só ocorrem nos dois últimos níveis da árvore.

Defina  $nn_k$  e  $ne_k$  como o número de nós e o número de folhas de erro, respectivamente, tendo profundidade  $k$ , em uma árvore par  $T$ .

Uma *árvore  $\alpha$ -par*,  $0 \leq \alpha \leq 1$ , é uma árvore par com a seguinte propriedade:

para todo  $k$ , se existirem folhas de codificação com profundidade  $k$ , então

$$ne_k \geq \lceil \frac{\alpha \cdot nn_k}{2} \rceil.$$

Com esta definição, podemos agora ter folhas de erro próximas à raiz de uma árvore par, fato que não acontece em geral com árvores pares ótimas, porque as folhas de erro situam-se sempre nos dois últimos níveis da árvore.

Em particular, quando  $\alpha > 0$ , sempre teremos folhas de erro com a mesma profundidade de qualquer folha de codificação. Conseqüentemente, aumentamos o parâmetro  $Pt_e$ , ou seja, o poder de detecção de erros da árvore.

Deve ser notado que, se considerarmos  $\alpha = 0$ , estamos exatamente na situação anterior, de árvores pares. Portanto, árvores  $\alpha$ -pares ótimas incluem todas as árvores pares ótimas, mas permitem outros tipos de árvores ótimas.

Como as árvores  $\alpha$ -pares são árvores estritamente binárias, o número de nós de profundidade  $k$ , com paridade ímpar,  $no_k$ , é exatamente metade de todos os nós com profundidade  $k$ ,  $nn_k$ , para  $k > 0$ . Conseqüentemente, a definição de árvores  $\alpha$ -pares pode, alternativamente, ser escrita como:

para todo  $k$ , se existirem folhas de codificação com profundidade  $k$ , então

$$ne_k \geq \lceil \alpha \cdot no_k \rceil.$$

A Figura 7.1 mostra três árvores  $\alpha$ -pares ótimas ( $\alpha = 0$ ,  $0.5$  e  $1$ ) para um mesmo conjunto de cinco símbolos,  $\{s_1, s_2, s_3, s_4, s_5\}$ , com probabilidades  $\{0.44, 0.22, 0.15, 0.11, 0.08\}$ , respectivamente. Na parte esquerda superior, temos uma árvore  $0$ -par ótima, cujo custo é  $2.17$ ; na parte superior direita, uma árvore  $0.5$ -par ótima, cujo custo é  $2.76$  e, na parte inferior, uma árvore  $1$ -par ótima, com custo  $3.38$ .

Na segunda árvore, para  $\alpha = 0.5$ , o número mínimo de folhas de erros com profundidades  $2$  e  $4$  é  $\lceil \frac{0.5 \cdot 4}{2} \rceil = \lceil 0.5 \cdot 2 \rceil = 1$ . Para profundidade  $5$ , o número mínimo é  $\lceil \frac{0.5 \cdot 2}{2} \rceil = \lceil 0.5 \cdot 1 \rceil = 1$ .

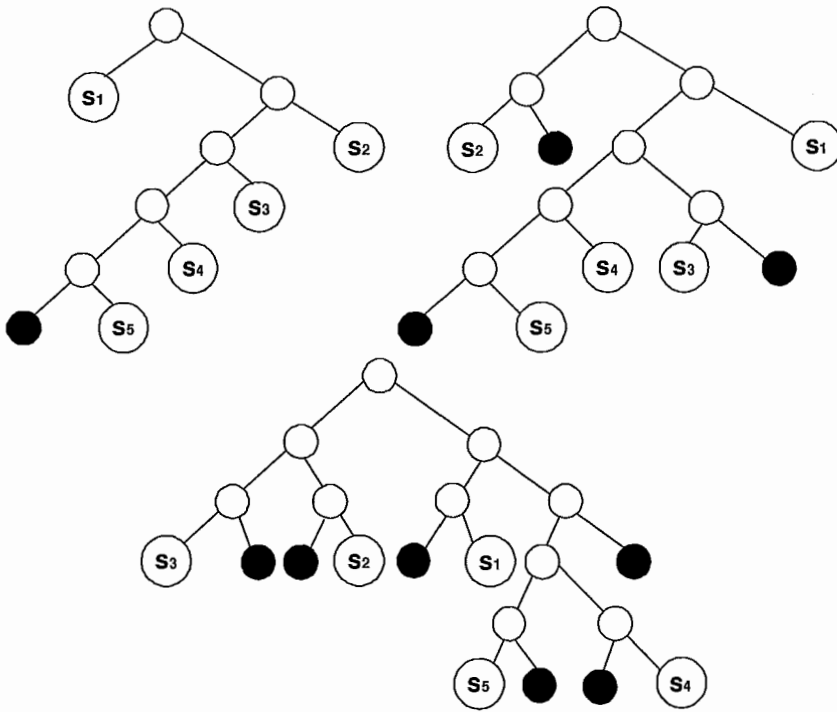


Figura 7.1: Exemplos de árvores 0-pares, 0.5-pares e 1-pares

A seguir, vamos caracterizar as árvores 1-pares ótimas, para a situação de probabilidades uniformes.

## 7.1 Árvores 1-pares cheias, subcheias e $k$ -completas

Inicialmente vamos definir três tipos especiais de árvores 1-pares.

Uma *árvore 1-par cheia*, de profundidade  $d$ , é uma árvore 1-par,  $T$ , contendo  $n = 2^{d-1}$  folhas de codificação, tal que todas as folhas têm profundidade  $d$ .

A Figura 7.2 mostra uma árvore 1-par cheia de profundidade 3.

Pode-se verificar facilmente as seguintes propriedades de uma árvore 1-par cheia,  $T$ :

a)  $T$  é uma árvore binária cheia, sendo que as folhas de paridade par são folhas de codificação e as folhas de paridade ímpar são folhas de erro. Temos  $2^{d-1}$  folhas de codificação e igual número de folhas de erro.

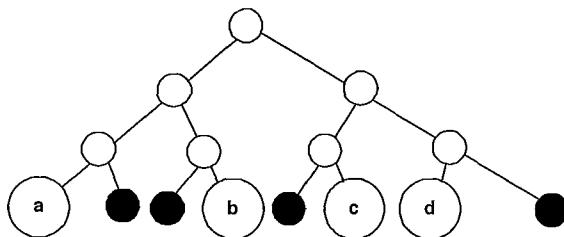


Figura 7.2: Árvore 1-par cheia de profundidade 3

b) a profundidade de  $T$  é  $d = \log n + 1$ .

Uma *árvore 1-par subcheia*, de profundidade  $d$ , com  $n$  folhas de codificação, é uma árvore 1-par,  $T$ , cujas folhas de codificação têm todas profundidade  $d$ , tal que  $2^{d-2} < n \leq 2^{d-1}$ .

A Figura 7.3 mostra uma árvore 1-par subcheia de profundidade 3, contendo 3 folhas de codificação.

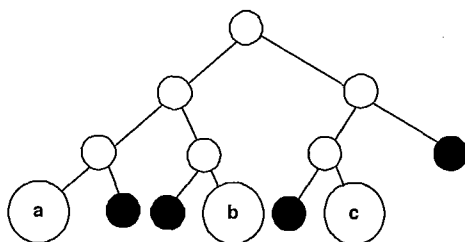


Figura 7.3: Árvore 1-par subcheia de profundidade 3, com  $n = 3$

Pode-se verificar facilmente as seguintes propriedades de uma árvore 1-par subcheia,  $T$ :

a) uma árvore 1-par cheia é um caso particular de árvore 1-par subcheia, contendo o número máximo de folhas de codificação possível.

b) a profundidade de  $T$  é  $d = \lceil \log n \rceil + 1$ .

c) sendo  $ne$  o número de folhas de erro de  $T$ , temos:

$d - 1 + 2^{d-2} \leq ne \leq 2^{d-1}$ . O valor mínimo para  $ne$  ocorre quando  $n = 2^{d-2} + 1$ , e o máximo, quando  $n = 2^{d-1}$ .

Uma *árvore 1-par  $k$ -completa*, de profundidade  $d$ , é uma árvore 1-par,  $T$ ,



cujas folhas de codificação estão distribuídas em dois níveis de profundidades  $d_1$  e  $d_2$ , e tal que:

- a)  $d = d_2 = d_1 + k$
- b) a subárvore induzida de profundidade  $d_1$  é uma árvore binária cheia.
- c) cada subárvore com raiz no nível  $d_1 + 1$  é uma árvore 1-par cheia de profundidade  $k$ .

A Figura 7.4 mostra uma árvore 1-par 2-completa, com  $d_1 = 3$  e contendo 5 folhas de codificação.

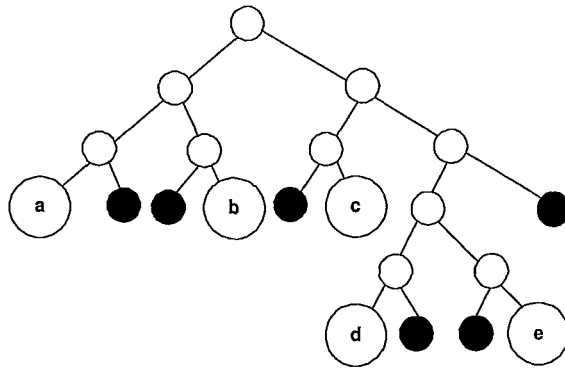


Figura 7.4: Árvore 1-par 2-completa de profundidade 5, com  $n = 5$

**Lema 7.1.1** *Em uma árvore 1-par 2-completa,  $T$ , com  $n$  folhas de codificação,  $d = \lceil \log n \rceil + 2$ .*

**Demonstração:** Sejam  $q_1$  o número de folhas de codificação de  $T$  com profundidade  $d - 2$  e  $q_2$  o número de folhas de codificação com profundidade  $d$ . Temos:

$$q_1 + q_2/2 = 2^{d-3} = \text{número de nós de paridade par com profundidade } d-2.$$

Como  $n = q_1 + q_2$ ,  $2^{d-3} > q_1 > 1$  e  $q_2 > 1$ , temos:

$$q_1 < 2^{d-3} < n < 2^{d-2} = 2q_1 + q_2 \Rightarrow d-3 < \log n < d-2 \Rightarrow d = \lceil \log n \rceil + 2. \quad \blacksquare$$

## 7.2 Árvores 1-pares ótimas para probabilidades uniformes

A partir deste ponto, consideraremos árvores 1-pares ótimas, para a situação de probabilidades uniformes.

O *Caminho de Codificação* de uma árvore 1-par,  $T$ , contendo  $n$  folhas de codificação é definido por  $P(T) = \sum_{i=1}^n d_i$ , onde  $d_i$  é a profundidade da folha de codificação  $i$ . O custo de  $T$  é definido por  $c(T) = \sum_{i=1}^n d_i f_i$ , onde  $d_i$  é a profundidade da folha de codificação  $i$  e  $f_i$  é a probabilidade associada a essa folha. Uma árvore 1-par  $T$  é ótima quando ela é a que tem o menor custo. Quando as probabilidades são uniformes, temos  $f_i = 1/n$ ,  $1 \leq i \leq n$ . Neste caso, a árvore ótima também possui o menor Caminho de Codificação, pois  $c(T) = P(T)/n$ . Por ser mais simples a definição deste último parâmetro, ele será a referência adotada na discussão seguinte sobre árvores 1-pares ótimas.

**Lema 7.2.1** *Em uma árvore 1-par subcheia,  $T$ , com  $n$  folhas de codificação,  $P(T) = n(\lceil \log n \rceil + 1)$ .*

**Demonstração:** Imediato, pois todas as  $n$  folhas de codificação têm profundidade  $\lceil \log n \rceil + 1$ . ■

**Lema 7.2.2** *Em uma árvore 1-par 2-completa,  $T$ , com  $n$  folhas de codificação, temos:  $P(T) = n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1}$ .*

**Demonstração:** Sejam  $q_1$  o número de folhas de codificação de  $T$  com profundidade  $d - 2$  e  $q_2$  o número de folhas de codificação com profundidade  $d$ .

Temos:

$$q_1 = 2^{d-3} - q_2/2 = 2^{d-3} - (n - q_1)/2 \Rightarrow q_1 = 2^{d-2} - n.$$

$$q_2 = n - (2^{d-2} - n) = 2n - 2^{d-2}. \text{ Logo:}$$

$$P(T) = q_1(d - 2) + q_2d = (2^{d-2} - n)(d - 2) + (2n - 2^{d-2})d$$

$$\text{Portanto, } P(T) = n(d + 2) - 2^{d-1} = n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1}. \quad \blacksquare$$

**Lema 7.2.3** *Em uma árvore 1-par ótima,  $T$ , não existem folhas de codificação em níveis consecutivos.*

**Demonstração:** Imediato porque, se por absurdo isso ocorresse, os nós pai das folhas do último nível teriam que ter paridade par. Nesse caso, essas folhas poderiam ser movidas para o nó pai e  $T$  não seria ótima, pois a nova árvore teria menor Caminho de Codificação. ■

Vamos definir duas operações em árvores 1-pares, necessárias aos teoremas posteriores: a ampliação de nível e a eliminação de folha.

Dada uma árvore 1-par  $T$ , onde todos nós de profundidade  $d_i$  e paridade ímpar sejam folhas de erro, a *ampliação do nível de profundidade  $d_i$*  é uma transformação de  $T$  para  $T'$ , da seguinte forma:

a) folhas de  $T$  com profundidade  $d_i$  são substituídas por uma subárvore consistindo de uma raiz e dois filhos.

b) na nova árvore  $T'$ , as folhas de paridade ímpar e profundidade  $d_i + 1$  são folhas de erro e, as de paridade par, folhas de codificação.

c) os demais nós são os mesmos de  $T$ .

A Figura 7.5 ilustra a ampliação do nível de profundidade 2 de uma árvore 1-par.

É imediato que a ampliação do nível de profundidade  $d_i$  de uma árvore par  $T$ , com  $ne_i$  folhas de erro e  $q_i$  folhas de codificação, todas com profundidade  $d_i$ , gera uma nova árvore par com  $ne_i$  novas folhas de codificação e  $q_i$  novas folhas de erro, todas com profundidade  $d_i + 1$ .

A *eliminação de folha de codificação* em uma árvore 1-par,  $T$ , é uma operação recursiva, descrita pelo seguinte algoritmo, cujo parâmetro  $f$  é a folha a ser eliminada.

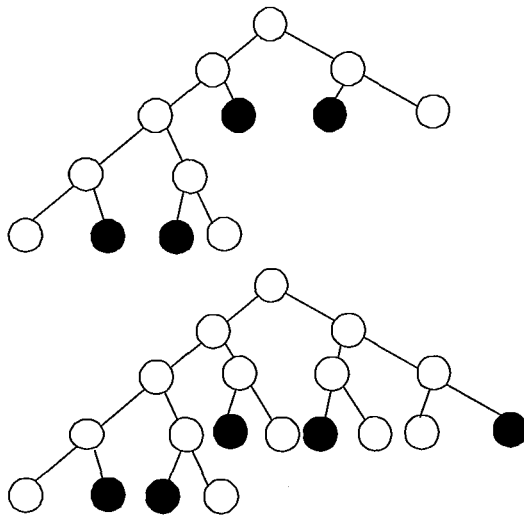


Figura 7.5: Ampliação do nível de profundidade 2 de árvore 1-par

### Algoritmo *Elimina(f)*

**Início:**

Transformar  $f$  em folha de erro;

Se ( $f$  é raiz) Então

$T \leftarrow$  Árvore Nula;

Senão

$p \leftarrow$  nó pai de  $f$  em  $T$ ;  $q \leftarrow$  irmão de  $f$  em  $T$ ;

Se ( $q$  é folha de erro) Então

Retirar  $f$  e  $q$  da árvore;

Elimina( $p$ );

Senão Se a paridade de  $q$  é ímpar Então

Trocar entre si  $f$  e  $q$  na árvore;

Trocar entre si os filhos de  $q$  na árvore;

**Fim;**

A idéia do algoritmo é dada a seguir. O processo de eliminação de uma folha  $f$  pode ser propagado até a raiz da árvore, através de deleções sucessivas de folhas. Quando  $f$  é a raiz, a árvore é transformada em árvore nula. Quando  $f$  não é raiz há dois casos. Se o nó irmão de  $f$ ,  $q$ , é também uma folha de erro,  $f$  e  $q$  são retiradas da árvore e, recursivamente, elimina-se  $p$ , nó pai de ambas.

Quando  $q$  não é folha de erro e tem paridade ímpar, então  $q$  é a raiz de uma subárvore e tem-se que acertar paridades. Para tanto, trocam-se entre si  $f$  e  $q$  e, adicionalmente, trocam-se também entre si as duas subárvores de  $q$  e o processo termina.

Portanto, o resultado da eliminação de uma folha de codificação em uma árvore 1-par, como descrito, resulta sempre numa árvore 1-par. Quando a árvore é ótima, o resultado não é necessariamente outra árvore ótima. Esse algoritmo também poderia ser aplicado a qualquer árvore  $\alpha$ -par.

As Figuras 7.6 a 7.8 ilustram o processo descrito. Por exemplo, a primeira eliminação, relativa à folha de codificação  $s_1$ , ocorre da seguinte forma: como o irmão de  $s_1$  é uma folha de erro, ambas as folhas são retiradas da árvore e o pai de ambas, digamos  $p$ , é transformado em folha de erro. Em seguida  $p$  vai ser recursivamente eliminada. Como o irmão de  $p$  não é folha de erro (é um nó interno), então fazem-se duas trocas na árvore:  $p$  é trocado com seu irmão e os filhos do irmão são trocados entre si. O resultado final é mostrado na Figura 7.6.

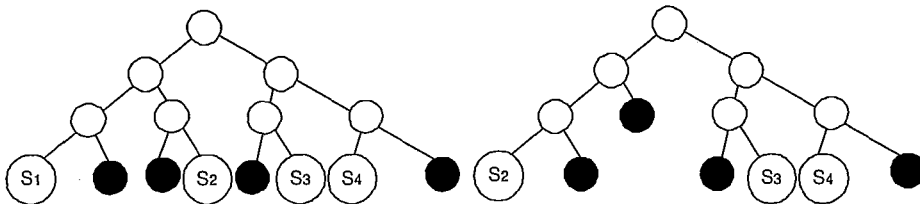


Figura 7.6: Eliminação da folha de codificação  $s_1$  na árvore 1-par

As figuras 7.7 e 7.8 ilustram, respectivamente, o resultado da eliminação das folhas  $s_4$  e  $s_2$ .

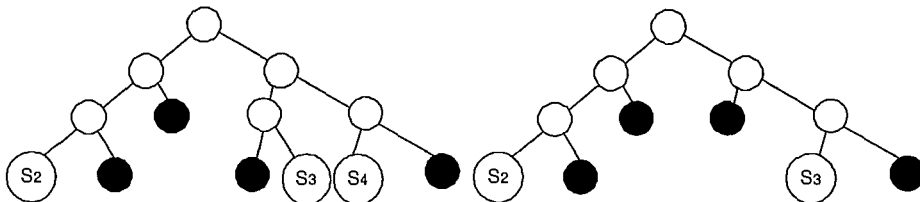


Figura 7.7: Eliminação da folha de codificação  $s_4$  na árvore 1-par

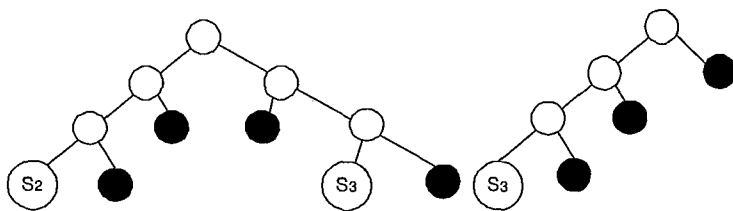


Figura 7.8: Eliminação da folha de codificação  $s_2$  na árvore 1-par

**Teorema 7.2.4** *Em uma árvore 1-par ótima,  $T$ , as folhas de codificação têm, no máximo, duas profundidades distintas.*

**Demonstração:** Seja  $T$  uma árvore 1-par ótima, tendo  $n$  folhas de codificação, com  $q_1, \dots, q_k, k > 2$  folhas de codificação com profundidades  $d_1, \dots, d_k$  respectivas e todas as profundidades distintas. Pelo Lema 7.2.3,  $d_{i+1} - d_i \geq 2, 1 \leq i < k$ . Vamos mostrar que isso é absurdo, pela exibição de uma árvore 1-par,  $T'$ , com  $P(T') < P(T)$ .

Inicialmente podemos constatar que a subárvore induzida a partir da raiz, com profundidade  $d_1$  é cheia pois, se não fosse, construiríamos  $T'$  a partir de  $T$ , acrescentando mais uma folha de codificação com profundidade  $d_1$  nessa subárvore e eliminando uma folha de codificação com profundidade  $d_k$ . Então teríamos  $P(T') < P(T)$ , absurdo. Por argumento análogo, concluímos que todas as subárvores induzidas entre dois níveis consecutivos  $niv_i$  e  $niv_{i+1}, 1 \leq i < k-1$  onde haja folhas de codificação, são cheias. Só podemos ter subárvores induzidas não cheias no caso das subárvores induzidas entre os dois últimos níveis.

Vamos examinar dois casos:

a)  $s = \sum_{i=2}^k q_i \geq q_1$ .

Neste caso, construiríamos  $T''$  a partir de  $T$ , ampliando o nível de profundidade  $d_1$ . Sendo  $ne_1$  o número de folhas de erro com profundidade  $d_1$  em  $T$ , passamos a ter  $ne_1 + q_1 > 2q_1$  folhas de codificação com profundidade  $d_1 + 1$  em  $T''$ . Em seguida, eliminamos  $q_1$  folhas dos últimos níveis de  $T''$  e também  $ne_1 - q_1$  folhas de codificação recém criadas com profundidade  $d_1 + 1$ , restando  $2q_1$  folhas de codificação com essa profundidade. Temos:

$P(T') = P(T) + q_1 - \sum_{i=1}^{q_1} d_i - d_1 - 1$ , já que  $q_1$  folhas tiveram seu nível ampliado de 1 e  $q_1$  outras tiveram seu nível reduzido de pelo menos 1. Pela hipótese, pelo menos uma dessas  $q_1$  folhas teve seu nível reduzido de mais de 1. Logo,  $P(T') < P(T) + q_1 - q_1$ . Ou seja,  $P(T') < P(T)$ , absurdo.

b)  $s = \sum_{i=2}^k q_i < q_1$ . Temos que examinar 3 subcasos.

b.1)  $d_2 = d_1 + 2$ .

Construiríamos  $T'$  a partir de  $T$  da seguinte forma: transformamos  $q_k - 1$  folhas de profundidade  $d_1$  em subárvores cheias de altura 2. Passamos a ter novas  $2(q_k - 1)$  folhas de codificação e igual número de folhas de erro com profundidade  $d_1 + 2$ . Eliminamos  $q_k - 1$  folhas de codificação com profundidade  $d_k$ . A subárvore com raiz na profundidade  $d_k - 1$  que tem uma única folha de codificação restante com profundidade  $d_k$  pode ser transformada, eliminando-se todos os descendentes dessa subárvore, ficando apenas a raiz. Logo,

$$P(T') = P(T) + 2(q_k - 1) - (d_k - (d_1 + 2))(q_k - 1) - (d_k - d_{k-1}),$$

já que  $q_k - 1$  folhas que tinham profundidade  $d_1$  passaram a ter profundidade  $d_1 + 2$ ,  $q_k - 1$  folhas de codificação que tinham profundidade  $d_k$  passaram a ter profundidade  $d_1 + 2$  e uma folha de codificação que tinha profundidade  $d_k$  passou a ter profundidade  $d_k - 1$ . Então

$$P(T') = P(T) - (q_k - 1)(d_k - d_1 - 4) - (d_k - d_{k-1}).$$

Como  $(d_k - d_1) \geq 4$  e  $d_k > d_{k-1}$  temos  $P(T') < P(T)$ , absurdo.

b.2)  $d_2 = d_1 + 3$ .

Construiríamos  $T'$  a partir de  $T$  da seguinte forma: transformamos 1 folha de profundidade  $d_1$  em uma subárvore subcheia de profundidade 3, com 3 folhas de codificação. Em seguida, eliminamos 2 folhas de codificação com profundidade  $d_k$ . Temos:

$$P(T') = P(T) + 3 - 2(d_k - (d_1 + 3)) = P(T) - 2(d_k - d_1) + 9.$$

Como, pelo Lema 7.2.3,  $(d_k - d_1) \geq 5$ , temos  $P(T') < P(T)$ , absurdo.

b.3)  $d_2 \geq d_1 + 4$ .

Seja  $T'$  a árvore 2-completa para  $n$ . Vamos mostrar que  $T'$  tem custo menor que o custo de  $T$ . Consideremos  $r = \sum_{i=2}^k q_i$ . Temos  $2^{d_1-1} < n = q_1 + r < 2q_1 < 2^{d_1}$ . Portanto,  $\lceil \log n \rceil = d_1$ . Temos ainda:

$$P(T') = n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1} = (q_1 + r)(d_1 + 4) - 2^{d_1+1} = \\ q_1 d_1 + r(d_1 + 4) - (2^{d_1+1} - 4q_1).$$

$$P(T) = q_1 d_1 + \sum_{i=2}^k q_i d_i > q_1 d_1 + r(d_1 + 4) = P(T') + (2^{d_1+1} - 4q_1) > P(T'),$$

pois  $2^{d_1-1} > q_1$ , uma vez que existe pelo menos um nó intermediário com profundidade  $d_1$ . Isto é absurdo, pois  $T$  é ótima.

Logo,  $T$  não pode ter folhas de codificação com mais de duas profundidades distintas. ■

**Teorema 7.2.5** *Em uma árvore 1-par ótima,  $T$ , se as folhas de codificação têm duas profundidades distintas, a diferença entre as profundidades é 2 ou 3 e  $T$  é equivalente a uma árvore 1-par 2-completa.*

**Demonstração:**

Seja  $T$  uma árvore 1-par ótima com  $n$  folhas de codificação, tendo as folhas duas profundidades distintas,  $d_1$  e  $d_2$ . A subárvore induzida de profundidade  $d_1$  é cheia. Suponhamos  $d_2 \geq d_1 + 4$ . Vamos mostrar que então podemos construir uma árvore 1-par,  $T'$ , com  $P(T') < P(T)$ . Temos que examinar dois casos.

a)  $q_2 \geq q_1$ .

Neste caso, construiríamos  $T''$  a partir de  $T$ , ampliando o nível de profundidade  $d_1$ . Sendo  $ne_1$  o número de folhas de erro com profundidade  $d_1$ , passamos a ter  $ne_1 + q_1$  folhas de codificação com profundidade  $d_2$ . Em seguida,



eliminamos  $q_1$  folhas de codificação com profundidade  $d_2$  e  $ne_1 - q_1$  folhas de codificação com profundidade  $d_1 + 1$ , obtendo  $T'$ . Temos:

$P(T') = P(T) + q_1 - q_1(d_2 - (d_1 + 1)) = P(T) - q_1(d_2 - d_1 - 2)$ , já que  $q_1$  folhas tiveram sua profundidade aumentada de 1 e  $q_1$  outras tiveram sua profundidade mudada de  $d_2$  para  $d_1 + 1$ . Como, pela hipótese,  $d_2 > d_1 + 2$ , segue-se que  $P(T') < P(T)$ , absurdo.

b)  $q_2 < q_1$ .

Temos:  $2^{d_1-1} < n = q_1 + q_2 < 2q_1 < 2^{d_1}$ . Portanto,  $\lceil \log n \rceil = d_1$ . Se construirmos uma árvore 1-par 2-completa,  $T'$ , teremos:

$$P(T') = n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1} = n(d_1 + 4) - 2^{d_1+1}.$$

$$P(T) = (n - q_2)d_1 + q_2d_2 = nd_1 + q_2(d_2 - d_1).$$

$$P(T) = c(T') + q_2(d_2 - d_1) - 4(n - 2^{d_1-1}).$$

Como  $(n - 2^{d_1-1}) < q_2$  e  $(d_2 - d_1) \geq 4$ , temos  $P(T) > P(T')$ , absurdo.

Logo,  $d_2 = d_1 + 2$  ou  $d_2 = d_1 + 3$ . Vamos verificar, a seguir, que é possível ter-se a árvore 1-par  $T'$  equivalente a  $T$ , tal que  $T'$  seja 2-completa. Temos que examinar 2 casos.

c)  $d_2 = d_1 + 2$ .

Neste caso,  $T$  já é uma árvore 2-completa pois se, em alguma subárvore com raiz no nível de profundidade  $d_1$ , houvesse apenas uma folha, esta folha poderia ser removida para o nó pai, decrescendo o custo de  $T$ , absurdo.

d)  $d_2 = d_1 + 3$ .

Pelo mesmo argumento usado no subcaso a), devemos ter  $q_2 < q_1$ . Se construirmos a árvore 2-completa,  $T'$ , teremos  $d(T') = d_1 + 1$ , conforme mostrado no subcaso b). Então:

$$P(T') = n(d_1 + 4) - 2^{d_1+1} = (q_1 + q_2)(d_1 + 4) - 2^{d_1+1}. \text{ E,}$$

$$P(T) = q_1d_1 + q_2(d_1 + 3).$$

Para que exista a igualdade  $P(T') = P(T)$ , devemos ter:

$$(q_1 + q_2)(d_1 + 4) - 2^{d_1+1} = q_1d_1 + q_2(d_1 + 3) \Rightarrow 2^{d_1+1} = 4q_1 + q_2 \quad (1)$$

A subárvore induzida de profundidade  $d_1$  é cheia. Seja  $x$  o número de nós pares intermediários com essa profundidade. Então:

$$q_1 + x = 2^{d_1-1} \quad (2)$$

Podemos considerar que, em  $T$ , no máximo uma subárvore enraizada no nível de profundidade  $d_1$  tenha menos que 4 folhas de codificação. Portanto:

$$q_2 \leq 4x \quad (3)$$

De (1), (2) e (3), temos:

$$2^{d_1+1} = 4q_1 + q_2 \leq 4q_1 + 4x = 2^{d_1+1}.$$

A única solução para este sistema é  $q_2 = 4x$ , o que significa que, numa árvore 1-par ótima com folhas tendo duas profundidades e com  $d_2 = d_1 + 3$ , as subárvores enraizadas no nível de profundidade  $d_1$  são cheias, contendo 4 folhas cada uma.

Concluindo, se  $T$  é ótima e tem folhas com duas profundidades distintas, a diferença de profundidade é 2 ou 3 e sempre existe uma árvore equivalente ótima 2-completa. ■

**Teorema 7.2.6** *Uma árvore 1-par ótima, com  $n$  folhas de codificação, probabilidades uniformes, é subcheia, se  $n > \frac{2^{\lceil \log n \rceil + 1}}{3}$ , ou 2-completa, se  $n < \frac{2^{\lceil \log n \rceil + 1}}{3}$ . Adicionalmente, pode também ser 3-completa, quando  $n$  é da forma  $n = 2^{\lceil \log n \rceil - 1} + 3k$ ,  $k$  inteiro.*

### Demonstração:

Seja  $T$  uma árvore 1-par ótima com  $n$  folhas de codificação. Pelo Teorema 7.2.5, ou  $T$  é subcheia ou equivalente a uma árvore 1-par 2-completa. Sejam  $T_s$  a árvore 1-par subcheia com  $n$  folhas e  $T_2$  a árvore 1-par 2-completa com o mesmo número de folhas. Consideremos 2 casos.

a)  $n > \frac{2^{\lceil \log n \rceil + 1}}{3}$ .

Temos:

$$P(T_s) = n(\lceil \log n \rceil + 1); \quad P(T_2) = n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1}.$$

Então:

$$P(T_s) - P(T_2) = 2^{\lceil \log n \rceil + 1} - 3n < 2^{\lceil \log n \rceil + 1} - 3 \frac{2^{\lceil \log n \rceil + 1}}{3} = 0.$$

Ou seja:  $P(T_s) < P(T_2)$ . Logo,  $T$  é uma árvore 1-par subcheia.

$$\text{b) } n < \frac{2^{\lceil \log n \rceil + 1}}{3}.$$

Temos:

$$P(T_s) - P(T_2) = 2^{\lceil \log n \rceil + 1} - 3n > 2^{\lceil \log n \rceil + 1} - 3 \frac{2^{\lceil \log n \rceil + 1}}{3} = 0.$$

Ou seja,  $P(T_2) < P(T_s)$ . Logo,  $T$  é equivalente a  $T_2$ , uma árvore 1-par 2-completa. Em algumas situações, a árvore pode também ser 3-completa. Vejamos as condições para isso ocorrer. Sejam  $q_1$  e  $q_2$  o número de folhas de codificações com profundidade  $d_1$  e  $d_1 + 3$ . Temos:

$$q_1 + q_2/4 = 2^{d_1 - 1}.$$

Isto implica  $4q_1 + 4q_2 = 2^{d_1 + 1} + 3q_2$ , ou seja:  $n = 2^{d_1 - 1} + \frac{3q_2}{4}$ . Como  $d_1 = \lceil \log n \rceil$  e  $q_2$  é múltiplo de 4, temos:

$$n = 2^{\lceil \log n \rceil - 1} + 3k, \text{ onde } k \text{ é o inteiro } q_2/4. \quad \blacksquare$$

A Tabela 7.1 ilustra o Teorema 7.2.6. São apresentados os seguintes parâmetros para árvores 1-pares ótimas,  $T$ , no caso de probabilidades uniformes:

$n$  = número de folhas de codificação,

$Tipo$  = se  $T$  é cheia, subcheia, 2-completa ou 3-completa,

$P(T)$  = caminho de codificação de  $T$ ,

$c(T)$  = custo de  $T$ ,

$d$  = profundidade de  $T$ ,

$q_1$  = número de folhas de codificação com menor profundidade,

$q_2$  = número de folhas de codificação com maior profundidade, nos casos em que  $T$  é 2-completa ou 3-completa.

Por exemplo, quando  $n = 88$ , temos  $2^{\lceil \log 88 \rceil + 1} = 256$ . Como  $88 > \frac{256}{3}$ , a árvore ótima é subcheia. Para  $n = 73$ , temos  $2^{\lceil \log 73 \rceil + 1} = 256$ . Como  $73 < \frac{256}{3}$ , a árvore ótima é 2-completa. Adicionalmente, como  $73 = 2^6 + 3.3$ , a árvore ótima também pode ser 3-completa.

Podemos ainda observar que, nos casos em que a árvore ótima é 2-completa, temos  $q_2 < q_1$ ; quando a árvore é 3-completa,  $q_2 < q_1/2$ .

$n$	Tipo	P(T)	$c(T)$	$d$	$q_1/q_2$	$n$	Tipo	P(T)	$c(T)$	$d$	$q_1/q_2$
60	subcheia	420	7	7	60/0	75	2-completa	569	7.59	9	53/22
61	subcheia	427	7	7	61/0	76	2-completa	580	7.63	9	52/24
62	subcheia	434	7	7	62/0	<b>76</b>	<b>3-completa</b>	580	7.63	10	60/16
63	subcheia	441	7	7	63/0	77	2-completa	591	7.68	9	51/26
64	cheia	448	7	7	64/0	78	2-completa	602	7.72	9	50/28
65	2-completa	459	7.06	9	63/2	79	2-completa	613	7.76	9	49/30
66	2-completa	470	7.12	9	62/4	<b>79</b>	<b>3-completa</b>	613	7.76	10	59/20
67	2-completa	481	7.18	9	61/6	80	2-completa	624	7.80	9	48/32
<b>67</b>	<b>3-completa</b>	481	7.18	10	63/4	81	2-completa	635	7.84	9	47/34
68	2-completa	492	7.24	9	60/8	82	2-completa	646	7.88	9	46/36
69	2-completa	503	7.29	9	59/10	<b>82</b>	<b>3-completa</b>	646	7.88	10	58/24
70	2-completa	514	7.34	9	58/12	83	2-completa	657	7.92	9	45/38
<b>70</b>	<b>3-completa</b>	514	7.34	10	62/8	84	2-completa	668	7.95	9	44/40
71	2-completa	525	7.39	9	57/14	85	2-completa	679	7.99	9	43/42
72	2-completa	536	7.44	9	56/16	<b>85</b>	<b>3-completa</b>	679	7.99	10	57/28
73	2-completa	547	7.49	9	55/18	86	subcheia	688	8	8	86/0
<b>73</b>	<b>3-completa</b>	547	7.49	10	61/12	87	subcheia	696	8	8	87/0
74	2-completa	558	7.54	9	54/20	88	subcheia	704	8	8	88/0

Tabela 7.1: Custos e tipos de árvores 1-par ótimas para frequências uniformes

Outra particularidade a ser observada é a existência de inúmeras árvores com número de codificações distintos e mesmo custo. Por exemplo, todas as árvores ótimas que tenham número de folhas de codificação compreendido entre 86 e 128 têm custo 8.

### 7.3 Comparação entre árvores de Huffman e árvores $\alpha$ -pares ótimas para probabilidades uniformes

Nesta seção analisaremos as diferenças entre Caminhos de Codificação e Custos entre árvores 1-pares ótimas e árvores de Huffman e árvores 0-pares

ótimas. Em capítulos anteriores, a comparação entre as duas últimas já foi efetuada.

Na Tabela 7.2 são apresentadas as expressões para o cálculo do Caminho de Codificação e Custos de árvores de Huffman, árvores ótimas 0-pares e árvores ótimas 1-pares, em função do número de codificações,  $n$ . No caso das árvores ótimas 1-pares, são apresentadas duas expressões, dependendo do valor de  $n$ .

Tipo	Caminho de Codificação	Custo
Huffman	$n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}$	$\lceil \log n \rceil + 1 - \frac{2^{\lceil \log n \rceil}}{n}$
0-pares	$n(\lceil \log \frac{n}{3} \rceil + 3) - 3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}$	$\lceil \log \frac{n}{3} \rceil + 3 - \frac{3 \cdot 2^{\lceil \log \frac{n}{3} \rceil}}{n}$
1-pares ( $n < \frac{2^{\lceil \log n \rceil + 1}}{3}$ )	$n(\lceil \log n \rceil + 4) - 2^{\lceil \log n \rceil + 1}$	$\lceil \log n \rceil + 4 - \frac{2^{\lceil \log n \rceil + 1}}{n}$
1-pares ( $n > \frac{2^{\lceil \log n \rceil + 1}}{3}$ )	$n(\lceil \log n \rceil + 1)$	$\lceil \log n \rceil + 1$

Tabela 7.2: Expressões do Caminho de Codificação de árvores ótimas

Já vimos que a diferença de custo entre árvores ótimas 0-pares e árvores de Huffman varia no intervalo  $[\frac{1}{3}, \frac{1}{2}]$ , sendo máxima ( $\frac{1}{2}$ ) para  $n$  da forma  $n = 2^k$  e mínima ( $\frac{1}{3}$ ) para  $n$  da forma  $n = 3 \cdot 2^k$ .

As diferenças de custo entre árvores 1-pares ótimas e árvores de Huffman são dadas pelo teorema a seguir.

**Teorema 7.3.1** *A diferença de custos entre uma árvore 1-par ótima,  $T_{E_1}$  e uma árvore de Huffman,  $T_H$ , para  $n$  símbolos, com probabilidades uniformes, varia no intervalo  $[1, 1.5)$ , sendo mínima para  $n$  da forma  $n = 2^k$ ,  $k$  inteiro.*

**Demonstração:** Consideremos dois casos.

$$\text{Caso 1): } 2^{\lceil \log n \rceil - 1} + 1 \leq n < \frac{2^{\lceil \log n \rceil + 1}}{3}.$$

Considerando a Tabela 7.2, a diferença de custos é:

$$c(T_{E_1}) - c(T_H) = \lceil \log n \rceil + 4 - \frac{2^{\lceil \log n \rceil + 1}}{n} - \lceil \log n \rceil - 1 + \frac{2^{\lceil \log n \rceil}}{n} = 3 - \frac{2^{\lceil \log n \rceil}}{n}.$$

$$\text{Para } n = 2^{\lceil \log n \rceil - 1} + 1, \text{ temos } 3 - \frac{2^{\lceil \log n \rceil}}{n} = 3 - \frac{2(n-1)}{n} > 3 - \frac{2n}{n} = 1.$$

Quando  $n = \lfloor \frac{2^{\lceil \log n \rceil + 1}}{3} \rfloor$ ,  $n$  pode ser escrito como  $n = \frac{2^{\lceil \log n \rceil + 1} - a}{3}$ , onde  $a = 1$  se  $\lceil \log n \rceil$  é ímpar ou  $a = 2$ , se  $\lceil \log n \rceil$  é par. Neste caso, a diferença é dada por  $3 - \frac{2^{\lceil \log n \rceil}}{\frac{2^{\lceil \log n \rceil + 1} - a}{3}} = 3(1 - \frac{2^{\lceil \log n \rceil}}{2^{\lceil \log n \rceil + 1} - a}) < 3(1 - \frac{2^{\lceil \log n \rceil}}{2^{\lceil \log n \rceil + 1}}) = 1.5$ . E, uma vez que

a função diferença é crescente no intervalo  $[2^{\lceil \log n \rceil - 1} + 1, \lfloor \frac{2^{\lceil \log n \rceil + 1}}{3} \rfloor]$ , segue-se que  $1 < c(T_{E_1}) - c(T_H) < 1.5$ .

Caso 2):  $\frac{2^{\lceil \log n \rceil + 1}}{3} < n \leq 2^{\lceil \log n \rceil}$ .

Considerando a Tabela 7.2, a diferença de custos é:

$$c(T_{E_1}) - c(T_H) = \lceil \log n \rceil + 1 - \lceil \log n \rceil - 1 + \frac{2^{\lceil \log n \rceil}}{n} = \frac{2^{\lceil \log n \rceil}}{n}.$$

Quando  $n$  é da forma  $n = 2^{\lceil \log n \rceil}$ , a diferença se torna  $\frac{2^{\lceil \log n \rceil}}{2^{\lceil \log n \rceil}} = 1$ . Para  $n = \lfloor \frac{2^{\lceil \log n \rceil + 1}}{3} \rfloor$ ,  $n$  pode ser escrito como  $n = \frac{2^{\lceil \log n \rceil + 1} + a}{3}$ , onde  $a = 1$  se  $\lceil \log n \rceil$  é par ou  $a = 2$ , se  $\lceil \log n \rceil$  é ímpar. Neste caso, a diferença é dada por  $\frac{2^{\lceil \log n \rceil}}{\frac{2^{\lceil \log n \rceil + 1} + a}{3}} = 3 \frac{2^{\lceil \log n \rceil}}{2^{\lceil \log n \rceil + 1} + a} < 3 \frac{2^{\lceil \log n \rceil}}{2^{\lceil \log n \rceil + 1}} = 1.5$ . E, uma vez que a função diferença é decrescente no intervalo  $[\lfloor \frac{2^{\lceil \log n \rceil + 1}}{3} \rfloor, 2^{\lceil \log n \rceil}]$ , segue-se que  $1 \leq c(T_{E_1}) - c(T_H) < 1.5$ . ■

A Tabela 7.3 ilustra os parâmetros Caminho de Codificação e custos para valores de  $n =$  número de símbolos, variando de 10 a 10000.

Na próxima seção apresentamos um algoritmo para determinar árvores  $\alpha$ -pares ótimas, probabilidades arbitrárias. No final do capítulo discutiremos o caso em que  $\alpha = 1$ .

## 7.4 Árvores $\alpha$ -pares ótimas para probabilidades arbitrárias

Apresentamos, a seguir, um algoritmo exato para construir árvores  $\alpha$ -pares ótimas. Este algoritmo é uma adaptação simples do algoritmo para obter árvores pares ótimas do Capítulo 4.

Seja  $\mathcal{S} = \{s_1, \dots, s_n\}$  um conjunto de símbolos, cada  $s_i$  com probabilidade  $f_i$ , satisfazendo  $f_i \leq f_{i+1}$  e  $\alpha$  um número real,  $0 \leq \alpha \leq 1$ . Para  $m \leq n$ , denotaremos  $\mathcal{S}_m = \{s_1, \dots, s_m\}$ .

Nosso objetivo é encontrar um código  $\alpha$ -par  $\mathcal{C}$  para  $\mathcal{S}$ , de custo mínimo. Proporemos, entretanto, a solução para um problema ligeiramente mais geral.

Uma *floresta de  $\alpha$ -paridade*  $F$  para  $\mathcal{S}_m$  é um conjunto de  $q$  árvores pares

n	Huffman		0-Even		1-Even	
	$P(T_H)$	$c(T_H)$	$P(T_{E_0})$	$c(T_{E_0})$	$P(T_{E_1})$	$c(T_{E_1})$
8	24	3	28	3.5	32	4
10	34	3.4	38	3.8	48	4.8
11	39	3.55	43	3.91	55	5
12	44	3.67	48	4	60	5
16	64	4	72	4.5	80	5
21	94	4.47	102	4.86	125	5.95
22	100	4.55	108	4.91	132	6
24	112	4.67	120	5	144	6
32	160	5	176	5.5	192	6
48	272	5.67	288	6	336	7
64	384	6	416	6.5	448	7
100	672	6.72	708	7.08	800	8
192	1472	7.67	1536	8	1728	9
512	4608	9	4864	9.5	5120	10
682	6478	9.5	6734	9.87	7500	11
683	6489	9.5	6745	9.88	7513	11
768	7424	9.67	7680	10	7680	11
1000	9976	9.98	10464	10.46	11000	11
5000	61808	12.36	63856	12.77	68616	13.72
10000	133616	13.36	137712	13.77	147232	14.72

Tabela 7.3: Comparação de custos entre Huffman e árvores  $\alpha$ -pares

e  $q$  árvores ímpares, tal que as folhas de paridade par das árvores pares e as folhas de paridade ímpar das árvores ímpares correspondam aos símbolos de  $\mathcal{S}_m$ , para  $1 \leq q \leq m$ . Temos uma restrição adicional: se qualquer das  $q$  árvores pares for uma árvore trivial (constituída somente pela raiz), então pelo menos  $\lceil \alpha q \rceil$  árvores ímpares devem também ser triviais.

Definimos o *custo* de  $F$  como a soma dos custos de suas árvores. Dizemos que  $F$  é  $(\alpha, m, q)$ -ótima quando seu custo é o menor dentre todas as florestas de  $\alpha$ -paridade para  $\mathcal{S}_m$ , tendo  $q$  árvores pares e  $q$  árvores ímpares. Denotaremos por  $c(\alpha, m, q)$  o custo de uma floresta  $(\alpha, m, q)$ -ótima.

Inicialmente, definiremos a função

$$A_i = \begin{cases} \sum_{j=1}^i f_j, & \text{se } i > 0 \\ 0, & \text{caso contrário} \end{cases}$$

Em termos desta notação, a solução para nosso problema é uma árvore tendo como subárvores as árvores de uma floresta  $(\alpha, n, 1)$ -ótima. Seu custo é  $c(\alpha, n, 1) + A_n$ .

O próximo teorema descreve a computação de  $c(\alpha, m, q)$ .

**Teorema 7.4.1** *Sejam  $m, q$  inteiros tais que  $1 \leq m \leq n$  e  $m \geq q \geq 1$ .*

(1) *se  $m \leq q$  então  $c(\alpha, m, q) = 0$ .*

(2) *se  $m > q$  então*

$$c(\alpha, m, q) = \min \{ c(\alpha, m, 2q) + A_m, \min_{1 \leq i \leq \text{lim}} \{ c(\alpha, m-i, 2q - \lceil \alpha q \rceil - i) + A_{m-i} \} \},$$

$$\text{onde } \text{lim} = \begin{cases} q - 1, & \text{se } q - \lceil \alpha q \rceil = 0 \\ q, & \text{caso contrário} \end{cases}$$

**Demonstração:** Por indução, mostraremos que os casos (1) e (2) computam corretamente  $c(\alpha, m, q)$ , para  $1 \leq m \leq n$  e  $m \geq q \geq 1$ . Quando  $m = 1$ , o caso (1) implica  $c(\alpha, 1, q) = 0$ , que é correto, uma vez que o único símbolo está na raiz de uma árvore par. Temos também  $\lceil \alpha q \rceil$  raízes ímpares como folhas de erro e as árvores restantes são nulas. Para  $m > 1$ , seja  $F$  uma floresta  $(\alpha, m, q)$ -ótima para  $\mathcal{S}_m$ . Consideremos as alternativas.

(1)  $m \leq q$ : Neste caso,  $m$  árvores pares triviais de  $F$  correspondem aos símbolos  $s_1, \dots, s_m$ , respectivamente. As restantes  $q - m$  árvores pares são árvores nulas. Temos também  $\lceil \alpha q \rceil$  árvores triviais ímpares correspondendo a folhas de erro e  $q - \lceil \alpha q \rceil$  árvores ímpares nulas. Portanto,  $c(m, q) = 0$ .

(2)  $m > q$ : temos dois casos:

2.1) Se não tivermos símbolos como raízes de  $F$ , então estes símbolos estão na floresta  $F''$  formada pela remoção das raízes das árvores pertencentes a  $F$ .  $F''$  é uma floresta  $(\alpha, m, 2q)$ -ótima. Relativamente a  $F$ , todos os símbolos de  $F''$  têm a profundidade aumentada em 1. Como consequência,  $c(\alpha, m, q) = c(\alpha, m, 2q) + A_m$ .



2.2) Se tivermos  $i$  árvores pares triviais de  $F$ , correspondendo aos símbolos  $s_{m-i+1}, \dots, s_m$ , para algum  $i \in \{1, \dots, \text{lim}\}$ , os  $m - i$  símbolos restantes estão distribuídos na floresta  $F'$ , com  $q - i$  árvores pares e  $q - \lceil \alpha q \rceil$  árvores ímpares. Note que a floresta  $F''$  formada pela remoção das raízes das árvores de  $F'$  é uma floresta  $(\alpha, m - i, 2q - \lceil \alpha q \rceil - i)$ -ótima. Com relação a  $F$ , todos os símbolos de  $F''$  têm a profundidade aumentada em 1. Portanto, para cada  $i$ ,  $c(\alpha, m, q) = c(\alpha, m - i, 2q - \lceil \alpha q \rceil - i) + A_{m-i}$ . Se  $q - \lceil \alpha q \rceil = 0$  todas as árvores ímpares de  $F'$  são folhas de erro. Nesta situação,  $\text{lim}$  deve ser, no máximo,  $q - 1$ . Caso contrário, podemos ter  $\text{lim} = q$ . O valor mínimo para  $c(\alpha, m, q)$  é o mínimo de  $c(\alpha, m - i, 2q - \lceil \alpha q \rceil - i) + A_{m-i}$ ,  $1 \leq i \leq \text{lim}$ .

Finalmente, minimizamos os casos 2.1) e 2.2). ■

O Teorema 7.4.1 nos leva a um algoritmo de programação dinâmica para determinar  $c(\alpha, m, q)$ , para todo  $1 \leq m \leq n$  e  $m \geq q \geq 1$ .

Comece avaliando a função  $A_i$  para  $1 \leq i \leq n$ . O parâmetro  $m$  varia de forma crescente,  $1 \leq m \leq n$ . Para cada  $m$ , o primeiro custo a ser computado é  $c(\alpha, m, m)$ , que é 0, por (1). Ainda para cada  $m$ ,  $q$  varia decrescentemente,  $1 \leq q < m$ , e, para cada par  $(m, q)$ , compute  $c(\alpha, m, q)$  aplicando (2). A computação pára quando  $c(\alpha, n, 1)$  é calculado, dado que nosso objetivo é obter  $c(\alpha, n, 1) + A_n$ . Há  $O(n^2)$  subproblemas. A avaliação de cada um é feita em tempo constante, se for usada a equação (1), ou em tempo  $O(n)$ , quando a avaliação é feita por (2). Consequentemente, a complexidade em tempo é  $O(n^3)$ . Os requisitos de espaço são  $O(n^2)$ .

Há um outro algoritmo mais eficiente para a construção de árvores 1-pares ótimas, cuja complexidade é  $O(n^2 \log n)$ . A seguir apenas delinearemos a idéia do mesmo, visto que a sua importância é pequena pois realmente só se aplica a este caso específico de árvores  $\alpha$ -pares.

A construção de uma floresta  $(1, m, q)$  ótima pode ser baseada na seguinte recorrência:

$$c(1, m, q) = \begin{cases} 0, & \text{se } m \leq q \\ \min_{0 \leq k \leq \lceil \log \frac{m}{q} \rceil} \{c(1, m-1, 2^k q - 1) + kA_m\}, & \text{se } m > q. \end{cases}$$

Para construir uma floresta  $F(1, m, q)$ -ótima, quando  $m \leq q$ , basta colocar  $m$  folhas de codificação em  $m$  raízes e fazer as demais  $q - m$  raízes serem folhas de erro. Quando  $m > q$ , então procura-se a posição para a folha de maior probabilidade que conduza a um custo total mínimo. A profundidade dessa folha é expressa na recorrência pelo valor  $k$ . Como essa folha é a de menor profundidade, então todas as subárvores de  $F$  são cheias até a profundidade  $k$ . O menor valor de  $k$  é zero e o maior é dado por  $\lceil \log \frac{m}{q} \rceil$ , situação em que todas as demais folhas também têm a mesma profundidade.

Desta forma, o número de subproblemas a serem resolvidos é  $O(n^2)$  e, para cada um deles, há a necessidade de examinar no máximo  $O(\log n)$  outros subproblemas, o que leva à complexidade de  $O(n^2 \log n)$ .

## 7.5 Resultados Experimentais

Nesta seção, apresentamos resultados experimentais para a detecção de erros feita por árvores  $\alpha$ -pares.

Os resultados experimentais estão sumarizados nas Tabelas 7.4 a 7.9. As tabelas apresentam parâmetros de árvores  $\alpha$ -pares ótimas obtidas pelos algoritmos, para vários valores de  $n$ . Esses dados foram originados por alguns programas escritos em linguagem Pascal, rodando em um computador Pentium IV de 1.8 GHz e 256M RAM.

Na Tabela 7.4 são apresentados os custos  $c(T_0)$ ,  $c(T_{0.1})$ ,  $c(T_{0.5})$ ,  $c(T_1)$  de quatro grupos de árvores  $\alpha$ -pares ótimas ( $\alpha = 0, 0.1, 0.5$  and  $1$ , respectivamente) e o custo  $c(T_H)$  da árvore de Huffman correspondente, para probabilidades uniformes, com  $n$  variando de 200 a 2000. São também mostradas as diferenças relativas entre os custos das árvores  $\alpha$ -pares e o custo da árvore de Huffman.

$n$	$c(T_H)$ (A)	$c(T_0)$ (B)	$c(T_{0.1})$ (C)	$c(T_{0.5})$ (D)	$c(T_1)$ (E)	$\frac{B-A}{A}$ %	$\frac{C-A}{A}$ %	$\frac{D-A}{A}$ %	$\frac{E-A}{A}$ %
200	7.72	8.08	8.22	8.40	9.00	4.7	6.5	8.8	16.6
400	8.72	9.08	9.21	9.40	10.00	4.1	5.6	7.8	14.7
600	9.29	9.72	9.76	10.15	10.59	4.6	5.1	9.3	14.0
800	9.72	10.08	10.21	10.40	11.00	3.7	5.0	7.0	13.2
1000	9.98	10.46	10.52	10.85	11.00	4.8	5.4	8.7	10.2
1200	10.29	10.72	10.76	11.15	11.59	4.2	4.6	8.4	12.6
1400	10.54	10.90	10.95	11.27	12.00	3.4	3.9	6.9	13.9
1600	10.72	11.08	11.21	11.40	12.00	3.4	4.6	6.3	11.9
1800	10.86	11.29	11.41	11.58	12.00	4.0	4.6	6.6	10.5
2000	10.98	11.46	11.52	11.85	12.00	4.4	4.9	7.9	9.3

Tabela 7.4: Comparação entre custos para probabilidades uniformes

A Tabela 7.5 é similar à 7.4, sendo que, como única diferença, é que na Tabela 7.5, é considerada a distribuição Zipf de probabilidades.

Pelos dados das tabelas, confirmamos o fato de que o custo da árvore 0-par ótima é bem próximo do da árvore de Huffman e que, para um número grande de símbolos, a diferença relativa é, no máximo, de 5%. Considerando a distribuição Zipf de probabilidades, as diferenças relativas são desprezíveis.

Com relação à árvore 0.1-par ótima, todas as diferenças relativas de custo são aproximadamente 5%, sugerindo alguma estabilidade.

Dentro de cada uma das tabelas, as diferenças relativas de custo decrescem com  $n$  e aumentam com  $\alpha$ . Mas, quando movemos de probabilidades uniformes para probabilidades Zipf, temos um comportamento singular. Enquanto as diferenças são menores para  $\alpha = 0$ , elas são bastante maiores para  $\alpha = 0.5$  e  $\alpha = 1$ .

Para distribuição Zipf de probabilidades e valores grandes de  $n$ , os dados sugerem que, relativamente ao código de Huffman, é necessário um bit a mais para cada codificação na árvore 0.5-par ótima e 2 bits a mais para a árvore 1-par ótima.

$n$	$c(T_H)$ (A)	$c(T_0)$ (B)	$c(T_{0.1})$ (C)	$c(T_{0.5})$ (D)	$c(T_1)$ (E)	$\frac{B-A}{A}$ %	$\frac{C-A}{A}$ %	$\frac{D-A}{A}$ %	$\frac{E-A}{A}$ %
200	6.03	6.07	6.37	7.04	7.91	0.7	5.6	16.8	31.2
400	6.67	6.71	7.03	7.75	8.68	0.6	5.4	16.2	30.1
600	7.05	7.08	7.42	8.14	9.13	0.4	5.2	15.5	29.5
800	7.31	7.35	7.68	8.43	9.43	0.6	5.1	15.3	29.0
1000	7.52	7.55	7.89	8.65	9.66	0.4	4.9	15.0	28.5
1200	7.68	7.71	8.06	8.83	9.86	0.4	4.9	17.6	28.4
1400	7.82	7.85	8.20	8.98	10.02	0.4	4.9	16.2	28.1
1600	7.94	7.97	8.32	9.12	10.15	0.4	4.8	15.7	27.8
1800	8.04	8.08	8.43	9.24	10.27	0.5	4.9	15.3	27.7
2000	8.14	8.17	8.53	9.34	10.38	0.4	4.8	15.2	27.5

Tabela 7.5: Comparação entre custos para a distribuição Zipf de probabilidades

Quando temos probabilidades uniformes, as árvores 1-par ótimas têm a característica especial de apresentar o mesmo custo para diferentes números de símbolos. Por exemplo, de 700 a 1000 símbolos, o custo de todas as árvores é 11.

As Tabelas 7.6 a 7.9 referem-se a probabilidades de detecção de erros.

Na Tabela 7.6 apresentamos os parâmetros  $Pt_e$  ( $Pt_e(T_0)$ ,  $Pt_e(T_{0.1})$ ,  $Pt_e(T_{0.5})$ ,  $Pt_e(T_1)$ ) de quatro grupos de árvores  $\alpha$ -pares ótimas ( $\alpha = 0, 0.1, 0.5$  e  $1$ , respectivamente), construídas pelo algoritmo exato apresentado anteriormente, considerando distribuição uniforme de probabilidades. A Tabela 7.7 tem o mesmo tipo de dados, para a distribuição Zipf.

Com relação às Tabelas 7.4 e 7.5, observamos que quanto maior a diferença de custos para a árvore de Huffman, maior é o parâmetro  $Pt_e(T)$ .

Podemos observar, também, um fato muito importante. Para  $\alpha > 0$ , temos uma melhoria significativa no parâmetro  $Pt_e$ , até mesmo para  $\alpha = 0.1$ . Neste caso, a despeito do pequeno aumento nos custos, a expectativa é a de se ter uma melhoria considerável no poder de detecção de erros. Estes dados sugerem que a extensão de árvores pares seja bastante efetiva.

$n$	$Pt_e(T_0)$	$Pt_e(T_{0.1})$	$Pt_e(T_{0.5})$	$Pt_e(T_1)$
200	0.1406	0.1592	0.3672	0.6094
400	0.1406	0.1558	0.3672	0.6094
600	0.0840	0.2197	0.4570	0.5430
800	0.1406	0.1482	0.3672	0.6094
1000	0.2383	0.2686	0.3541	0.5117
1200	0.1162	0.2197	0.4570	0.5430
1400	0.1582	0.1726	0.4082	0.6582
1600	0.1406	0.1482	0.3672	0.6094
1800	0.1895	0.1351	0.3433	0.5605
2000	0.2383	0.2684	0.3541	0.5117

Tabela 7.6: Parâmetro  $Pt_e$  para árvores  $\alpha$ -pares, distribuição uniforme de probabilidades

$n$	$Pt_e(T_0)$	$Pt_e(T_{0.1})$	$Pt_e(T_{0.5})$	$Pt_e(T_1)$
200	0.0205	0.1470	0.4118	0.5862
400	0.0222	0.1533	0.4118	0.6290
600	0.0150	0.1496	0.4055	0.6582
800	0.0154	0.1389	0.4274	0.6288
1000	0.0121	0.1478	0.4276	0.6436
1200	0.0154	0.1439	0.4328	0.6582
1400	0.0186	0.1484	0.4502	0.6436
1600	0.0125	0.1635	0.4551	0.6508
1800	0.0107	0.1835	0.4627	0.6578
2000	0.0125	0.1804	0.4347	0.6649

Tabela 7.7: Parâmetro  $Pt_e$  para árvores  $\alpha$ -pares, distribuição Zipf de probabilidades

$b$	$Pt_e(T_0)$ (0.0121)	$Pt_e(T_{0.1})$ (0.1478)	$Pt_e(T_{0.5})$ (0.4276)	$Pt_e(T_1)$ (0.6436)
10	0.763182	0.865025	0.941755	0.961157
25	0.783146	0.933582	0.976614	0.984462
50	0.811596	0.966182	0.988307	0.992231
100	0.854546	0.983085	0.994154	0.996116
250	0.921295	0.993234	0.997661	0.998446
500	0.958772	0.996617	0.998831	0.999223
1000	0.979339	0.998309	0.999415	0.999616
2500	0.991736	0.999323	0.999766	0.999847
5000	0.995868	0.999662	0.999883	0.999922

Tabela 7.8: Probabilidades teóricas de detecção de erros de árvores  $\alpha$ -pares, para distribuição Zipf de probabilidades

$b$	$Pt_e(T_0)$ (0.0121)	$Pt_e(T_{0.1})$ (0.1478)	$Pt_e(T_{0.5})$ (0.4276)	$Pt_e(T_1)$ (0.6436)
10	0.750865	0.961900	0.993534	0.997296
25	0.751565	0.986765	0.997735	0.998861
50	0.799236	0.991531	0.998905	0.999464
100	0.850530	0.997088	0.999389	0.999776
250	0.936867	0.998259	0.999770	0.999912
500	0.962826	0.999368	0.999904	0.999948
1000	0.983665	0.999625	0.999945	0.999974
2500	0.995633	0.999863	0.999979	0.999991
5000	0.997992	0.999923	0.999993	0.999997

Tabela 7.9: Probabilidades experimentais de detecção de erros de árvores  $\alpha$ -pares, para distribuição Zipf de probabilidades

Nas Tabelas 7.8 e 7.9 são apresentadas probabilidades de detecção de erros para distribuição Zipf de probabilidades, em árvores  $\alpha$ -pares ótimas, constuídas para 1000 símbolos.

Apresentamos, na Tabela 7.8, valores teóricos calculados pelo modelo do Capítulo 6. Na Tabela 7.9, obtivemos dados experimentais, através de um processo com um número muito grande de testes (mais de 20 milhões).

A primeira coluna de ambas as tabelas contém o número médio de símbolos em uma mensagem,  $b$ , variando de 10 a 50000. As quatro colunas seguintes mostram as probabilidades de detecção de erros de quatro diferentes grupos de árvores  $\alpha$ -pares ótimas, tendo  $Pt_e = 0.0121, 0.1478, 0.4276, 0.6436$ , respectivamente. Estes números são exatamente aquelas da quinta linha da Tabela 7.7.

Podemos observar, uma vez mais, a relação próxima entre valores teóricos e experimentais correspondentes. Isso significa que o modelo utilizado para calcular probabilidades é consistente.

Podemos também observar que temos uma excelente probabilidade de detecção de erros por árvores  $\alpha$ -pares ótimas, tanto para  $\alpha = 0.5$  e 1. Até mesmo para mensagens com baixo número de símbolos, as probabilidades são superiores a 99.4%. Mas também temos uma boa probabilidade de detecção de erros para  $\alpha = 0.1$ , com pequeno aumento nos custos.

Os resultados acima sugerem que encontramos um boa alternativa para combinar compactação com detecção de erros. A questão é como ajustar o nível de detecção de erros desejado, com a escolha apropriada do parâmetro  $\alpha$  para cada situação.

# Capítulo 8

## Conclusões

As árvores pares, objeto deste estudo, integram compactação e detecção de erros e foram inspiradas no modelo apresentado por Hamming em 1980, as árvores Hamming-Huffman. Esta proposta não recebeu qualquer desenvolvimento posterior, continuando em aberto sua análise, bem como a de métodos para a geração de árvores Hamming-Huffman ótimas. Ao que parece, esse desafio envolve interessantes problemas combinatórios.

Dividimos a análise das árvores pares em duas vertentes: situações de probabilidades uniformes e situações de probabilidades arbitrárias.

Caracterizamos totalmente as árvores pares na situação de probabilidades uniformes, podendo seu reconhecimento ser feito com complexidade  $O(n)$  em tempo. Apresentamos, também, um método que possibilita gerar todas as árvores ótimas. A comparação das árvores pares ótimas com as árvores de Huffman mostrou que a diferença de custos varia na faixa  $[1/3, 1/2]$ .

Para a situação geral, de probabilidades arbitrárias, foi apresentado um algoritmo de programação dinâmica, com complexidade  $O(n^3)$  em tempo e  $O(n^2)$  em espaço. Fica o desafio de se construir um algoritmo mais eficiente, especialmente em termos de espaço. Entretanto, foram apresentadas duas heurísticas para a obtenção de árvores pares quase ótimas, cuja diferença de custo para as correspondentes árvores de Huffman é limitada por  $\min\{0.5, c(T_H)/6\}$ , onde  $c(T_H)$  é o custo da árvore de Huffman. Ou seja, temos uma aproximação muito



boa das árvores ótimas e, além disso, vê-se que o custo para acrescentar poder de detecção às árvores de Huffman é baixo. Estas heurísticas têm complexidade  $O(n \log n)$  em tempo e  $O(n)$  em espaço e, portanto, igualam-se ao método de Huffman.

Outra análise importante das árvores pares diz respeito à capacidade de detecção de erros dessas estruturas. Foi desenvolvido um modelo probabilístico que apresentou grande aderência a inúmeros resultados experimentais. Este modelo é baseado na distribuição dos nós de erro na árvore e no número de símbolos da mensagem codificada. Em termos médios, tem-se uma detecção de erros bastante significativa (acima de 99,4%), para mensagens com mais de 100 símbolos.

Foi ainda apresentada uma importante extensão às árvores pares, as árvores  $\alpha$ -pares, cuja idéia é permitir maior poder de detecção de erros, especialmente no caso de compressão de linguagem natural, onde a árvore par é muito próxima à árvore de Huffman e só tem um bom poder de detecção de erros para mensagens com grande número de símbolos.

Aqui novamente foi dado um tratamento especial à situação de probabilidades uniformes, para o caso limite de  $\alpha = 1$ . Aplicando uma caracterização apresentada para as árvores 1-pares ótimas, elas podem ser construídas através de um método cuja complexidade é  $O(n)$  em tempo. Comparativamente às árvores de Huffman, o aumento de custo das árvores 1-pares ótimas correspondentes varia no intervalo  $[1, 1.5)$ .

Para a construção de árvores  $\alpha$ -pares ótimas, na situação de probabilidades arbitrárias, foi adaptado o mesmo algoritmo usado para árvores pares. Portanto, essas árvores ótimas podem ser construídas com complexidade  $O(n^3)$  em tempo e  $O(n^2)$  em espaço. Mostrou-se que a complexidade em tempo pode ser melhorada para o caso especial  $\alpha = 1$ , baixando-a para  $O(n^2 \log n)$ . Entretanto, permanece em aberto o desenvolvimento de algoritmos mais eficientes para esta situação ou, alternativamente, boas heurísticas, como aquelas obtidas para as árvores pares.

Finalmente, foram apresentados vários resultados experimentais para árvores

$\alpha$ -pares, no tocante à sua capacidade de detecção de erros, tendo sido percebido o fato encorajador de que um pequeno acréscimo no parâmetro  $\alpha$  já pode trazer ganhos significativos em termos do poder de detecção de erros. Vale a pena citar que a capacidade de detecção de erros apresentou uma melhoria significativa em relação às árvores 0-pares, no caso em que as frequências dos símbolos seguem as probabilidades de Zipf, padrão aproximado das linguagens naturais. Permanece em aberto, entretanto, o desenvolvimento de um modelo para avaliação da detecção de erros, bem como um método para a escolha do parâmetro  $\alpha$  adequado a cada situação.

Para resumir, nesta dissertação tratamos da integração entre a compactação de dados de Huffman e a detecção de erros. Esta é uma área pouquíssimo explorada na literatura, não havendo referências que possibilitem a comparação dos resultados aqui obtidos com outros enfoques. Esperamos que o trabalho possa contribuir para a discussão do assunto, porque os dois temas, compactação e controle de erros, permanecem sendo de grande interesse, especialmente frente ao crescimento explosivo da Internet.

A seguir relacionamos os problemas em aberto mencionados, que podem ser objeto de trabalhos futuros. O primeiro problema é o que motivou esta tese.

**Problema 8.1** *Encontrar uma árvore Hamming-Huffman ótima, que detecta erros de 1 bit, no símbolo onde o erro ocorreu.*

Podemos imaginar uma generalização do problema anterior, considerando a detecção de mais erros, o que leva ao segundo problema.

**Problema 8.2** *Encontrar uma árvore Hamming-Huffman ótima, que detecta erros de  $k$  bits, no símbolo onde os erros ocorreram.*

O algoritmo para se criar árvores pares ótimas tem complexidade  $O(n^3)$  em tempo e  $O(n^2)$  em espaço. Isso pode ser melhorado? Em particular, existiria um algoritmo guloso para a situação? O próximo problema considera essas questões.

**Problema 8.3** *Melhorar a complexidade da criação de árvores pares ótimas, especialmente a complexidade de espaço.*

Em relação ao estudo das árvores  $\alpha$ -pares, as duas próximas questões ficaram em aberto.

**Problema 8.4** *Desenvolver um modelo para avaliação da capacidade de detecção de erros de árvores  $\alpha$ -pares, bem como um método para a escolha do parâmetro  $\alpha$  para cada situação.*

**Problema 8.5** *Desenvolver algoritmos aproximativos para a construção de árvores  $\alpha$ -pares.*

Finalmente, pode-se imaginar um passo a mais no objetivo da integração da compactação com o controle de erros, que é o de se considerar a correção de erros e não apenas a detecção.

**Problema 8.6** *Desenvolver métodos para a integração da compactação de dados através de árvores de Huffman com a correção de erros.*

# Bibliografia

- [1] J. Abrahams, Code and parse trees for lossless source encoding, *Communications in Information and Systems*, 1(2001), pp. 113-146.
- [2] R. Baeza-Yates e B. Ribeiro-Netto, *Modern Information Retrieval*, Addison Wesley, New York, 1999.
- [3] M. Buro, On the maximum length of Huffman Codes, *Inform. Processing Letters*, 45(1993), pp. 219-223.
- [4] M. Burrows e D.J.Wheeler, A block sorting data compression algorithm, *Technical Report SRC 124*, Digital Systems Research Center, 1994.
- [5] J.G. Cleary e I.H.Witten, Data compression using adaptative coding and partial string matching, *IEEE Transactions on Communications*, 32(1984), pp. 396-402.
- [6] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, 1990.
- [7] S. X. Descamps, *Block Error-Correcting codes*, Springer, 2003.
- [8] N. Faller. An adaptative method for data compression, *Proc. of the 7th Asilomar Conference on Circuits, Systems and Computers*, Naval Post-graduate School, Monterrey, Ca., pp. 593-597, 1973.
- [9] R. M. Fano, *Transmission of Information*, M.I.T. Press, Cambridge Mass., 1949.

- [10] R. G. Gallager, Variations on a theme by Huffman, *IEEE Transactions on Information Theory*, 24(1978), pp. 668-674.
- [11] E. N. Gilbert, E. F. Moore, Variable-length binary encodings, *Bell Systems Technical Journal*, 38(1959), pp. 933-967.
- [12] M. Gutman, Fixed-prefix encoding of the integers can be Huffman-optimal, *IEEE Transactions on Information Theory*, 36(1990), pp. 936-938.
- [13] R. W. Hamming, *Coding And Information Theory*, Prentice Hall, Englewood Cliffs, New Jersey, 1980.
- [14] A. Hefez e M. L. T. Villela, *Códigos Corretores de Erros*, IMPA, Rio de Janeiro - Brasil, 2002.
- [15] D. A. Huffman, A method for the construction of minimum redundancy codes, *Proceedings of the IRE*, 40(1951), pp. 1098-1101.
- [16] J. L. Szwarcfiter e L. Markenzon, *Estruturas de Dados e seus Algoritmos*, LTC Editora, Rio de Janeiro - Brasil, 1994.
- [17] D. E. Knuth, *The Art of Computer Programming, V.1: Fundamental Algorithms*, Addison Wesley, 1968.
- [18] D. E. Knuth, *The Art of Computer Programming, V.3: Sorting And Searching*, Addison Wesley, 2nd Edition, 1973.
- [19] D. E. Knuth, Dynamic Huffman Coding, *Journal of Algorithms*, 6(1985), pp. 163-180.
- [20] L. L. Larmore and D. S. Hirshberg, A fast algorithm for optimal length-limited Huffman codes, *JACM*, 37(1990), pp. 464-473.
- [21] D. A. Lelewer and D. S. Hirshberg, Data compression, *ACM Computing Surveys*, 19(1987), pp. 261-296.

- [22] R. L. Milidiú, E. S. Laber and A. A. Pessoa, Improved Analysis of the FGK Algorithm, *Journal of Algorithms*, Vol. 28, pp. 195-211, 1999.
- [23] R. L. Milidiú, E. S. Laber and A. A. Pessoa, A work efficient parallel algorithm for constructing Huffman codes, In: Data Compression Conference, 1999, Snowbird - Utah, *Proceedings of the DCC'1999*, pp. 277-286, 1999.
- [24] R. L. Milidiú and E. S. Laber, The Warm-up algorithm: a Lagrangean construction of length restricted Huffman codes, *Siam Journal on Computing*, 30(2000), pp. 1405-1426.
- [25] R. L. Milidiú and E. S. Laber, Improved bounds on the inefficiency of length restricted codes, *Algorithmica*, 31(2001), pp. 513-529.
- [26] A. Moffat e J. Katajainen, In-place calculation of minimum-redundancy codes, *Proc. of Workshop on Algorithms and Data Structures*, pp. 393-402, 1995.
- [27] E. Moura, G. Navarro, N. Ziviani e R. Baeza-Yates, Fast searching on compressed text allowing errors, *Proc. of the 21st International ACM SIGIR Conference on research and development in Information Retrieval*, pp. 298-306, 1998.
- [28] E. Moura, G. Navarro, N. Ziviani e R. Baeza-Yates, Fast and flexible word searching on compressed text, *ACM Transactions on Information Systems*, 18(2000), pp. 113-139.
- [29] R. Pasco, Source coding algorithms for fast data compression, *Ph.D. Thesis*, Stanford Univ., 1976.
- [30] A. A. Pessoa, Comunicação pessoal, 2004.
- [31] P. E. D. Pinto, F. Protti and J. L. Szwarcfiter, Data compression and error detection integration, *Cadernos do IME*, 15-1(2003), pp 45-52.

- [32] P. E. D. Pinto, F. Protti e J. L. Szwarcfiter, Árvores de detecção ímpar ótimas, *Anais do XXVI CNMAC - Congresso Nacional de Matemática Aplicada e Computacional*, 1, pp. 253, São José do Rio Preto, SP, 2003.
- [33] P. E. D. Pinto, F. Protti e J. L. Szwarcfiter, Compactação de dados com detecção de erros, *Anais do XXXV SBPO - Simpósio Brasileiro de Pesquisa Operacional*, pp. 2462-2472, Natal, RN, 2003.
- [34] P. E. D. Pinto, F. Protti and J. L. Szwarcfiter, A Huffman-based error detecting code, In: Celso C. Ribeiro and Simone L. Martins (Eds.), *Experimental and Efficient Algorithms - Proc. of the WEA 2004, Lecture Notes in Computer Science*, 3059, pp. 446-457, 2004.
- [35] P. E. D. Pinto, F. Protti and J. L. Szwarcfiter, Parity Codes, *Rairo Inf. Theor. Appl.*, 39(2005), pp. 263-278.
- [36] P. E. D. Pinto, G. C Silva, Análise comparativa de métodos de compactação de dados sem perda, *Cadernos do IME*, 17-2(2005).
- [37] P. E. D. Pinto, F. Protti and J. L. Szwarcfiter, *Exact and experimental algorithms for a Huffman-based error detecting code*, Submitted, 2006.
- [38] J. J. Rissanen, Generalized Kraft inequality and Arithmetic Coding, *IBM Journal of Research and Development*, 20(1976), pp.198-203.
- [39] B. Rudner, Construction of minimum-redundancy codes with optimal synchronizing property, *IEEE Trans. Of Inform. Theory*, 17(1971), pp.478-487.
- [40] D. Salomon, *Data Compression The Complete Reference*, Springer, 1998.
- [41] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann Pub, 2000.
- [42] E. S. Schwartz, An optimum encoding with minimal longest code and total number of digits, *Information and Control*, 7(1964), pp. 37-44.

- [43] E. S. Schwartz e B. Kallic, Generating a canonical prefix encoding, *CACM*, 7(1964), pp. 166-169.
- [44] A. S. Tanebaum, *Redes de Computadores*, Ed. Campus, Rio de Janeiro, 4a edição, 2003.
- [45] A. Turpin and A. Moffat, Practical length-limited coding for large alphabets, *Computer J.*, Vol. 38, No 5, pp. 339-347, 1995.
- [46] T. A. Welch, A technique for high-performance data compression, *IEEE Computer*, junho 1984, pp. 8-19.
- [47] I. H. Witten, A. Moffat e T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, 2nd Edition, Morgan Kaufmann, 1999.
- [48] R.H. M. Zaragoza, *The Art of Correct Coding*, John Wiley & Sons, 2002.
- [49] G. K. Zipf, *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, 1949.
- [50] J. Ziv e A. Lempel, A universal algorithm for data compression, *IEEE Transactions on Information Theory*, 23(1977), pp. 337-343.
- [51] J. Ziv e A. Lempel, Compression of individual sequences via variable-rate coding, *IEEE Transactions on Information Theory*, 24(1978), pp. 530-536.
- [52] N. Ziviani, E. Moura, G. Navarro e R. Baeza-Yates, Compression: a key for next-generation Text Retrieval Systems, *IEEE Computer*, 33(2000), pp. 37-44.
- [53] N. Ziviani, *Projeto de Algoritmos*, Thomson, São Paulo, 2a edição, 2004.