CÓDIGOS DE PREFIXO: ALGORITMOS E COTAS

Eduardo Sany Laber, laber@cos.ufrj.br
Ruy Luiz Milidiú, milidiu@inf.puc-rio.br

Departamento de Informática, PUC-Rio, Brazil
Rua Marquês de São Vicente 225, RDC, sala 514, FPLF
Gávea, Rio de Janeiro, CEP 22453-900, phone 5521-511-1942

Resumo

Os códigos de prefixo têm importância fundamental na compressão e transmissão de dados. Estes códigos também apresentam relações com problemas de busca. Nesta tese, apresentamos novos resultados estruturais e algorítmicos sobre a classe dos códigos de prefixo. Explicamos teoricamente as boas taxas de compressão observadas para alguns métodos utilizados na prática. Propomos também algoritmos eficientes para construção de códigos de prefixo ótimos e variantes. Os principais resultados aqui descritos são os seguintes: um novo algoritmo paralelo para construção de códigos de prefixos ótimos, uma cota superior para a perda de compressão introduzida pela restrição de comprimento nos códigos de prefixo, uma cota superior para a perda de otimalidade introduzida pela restrição de altura nas árvores alfabéticas de busca; um algoritmo aproximativo e linear para construção de códigos de prefixo com restrição de comprimento; um algoritmo aproximativo com complexidade $O(n \log n)$ para construção árvores alfabéticas de busca com restrição de altura; uma nova versão do algoritmo WARM-UP com complexidade fortemente polinomial; um algoritmo linear para reconhecer códigos de prefixo ótimos com restrição de comprimento; uma prova afirmativa da conjectura de Vitter sobre o desempenho dos códigos de Huffman dinâmicos construídos pelo algoritmo FGK (Faller, Gallager e Knuth).

Abstract

The prefix codes play an important role in data compression and data communication. These codes also present relation with search problems. In this thesis, we present new structural and algorithmic results concerning the prefix code class. We theoretically explain results related to the high compression rates of some methods that have been used for practical purposes. We also propose efficient algorithms for constructing optimal prefix codes and some variants. The major results presented in the thesis are: a new parallel algorithm for constructing optimal prefix codes; a sharp upper bound for the compression loss introduced due the usage of length restricted prefix codes; an upper bound on loss of optimality due to the usage of length restricted alphabetic search trees; an $O(n \log n)$ time approximative algorithm for constructing length restricted prefix code; a $O(n \log n)$ time approximative algorithm for constructing length restricted alphabetic search trees;; a strongly polinomial version for the WARM-UP algorithm; a linear time algorithm for recognizing optimal length restricted prefix codes; a proof for Vitter's conjecture about the performance of the Dynamic Huffman Codes constructed by FGK (Faller, Gallager and Knuth) algorithm.

1 Introdução

Nos computadores e nos meios de transmissão digitais, as informações são representadas através de códigos. Os códigos, no contexto da eletrônica digital, podem ser vistos como sequências de bits "0" e "1" que representam algum tipo de informação. O modo com que as informações são codificadas impacta profundamente o desempenho e a confiabilidade dos sistemas digitais.

As propriedades e características requeridas por estes códigos variam de aplicação para aplicação. Em aplicações de comércio eletrônico nas quais transações são realizadas mediante o envio de um número de cartão crédito, é desejável uma codificação segura. Neste caso, segurança implica que uma pessoa não envolvida na transação não consiga obter o número do cartão por mais que tenha acesso a codificação.

Quando se transmite dados em uma rede de computação, é desejável que as informações sejam codificadas de modo compacto, minimizando o tráfego na rede. Outra característica necessária é que a representação permita um mecanismo de detecção e correção de erros, evitando que as aplicações fiquem extremamente suscetíveis a ruídos na rede e outros imprevistos.

Nesta tese consideramos uma classe de códigos bastante importante: a classe dos códigos de prefixo. Dado um alfabeto de entrada $\Sigma = \{a_1, a_2, \ldots, a_n\}$, um código de prefixo binário e um código binário tal que a *palavra código* (cadeia de bits) associada a a_i não é prefixo da palavra código associada a a_j, para $i \neq j$. Na tabela abaixo C_1 é um código de prefixo. Como 01 é prefixo de 010, C_2 não é um código de prefixo,

alfabeto	C_1	C_2
a_1	010	010
a_2	00	01
a_3	101	10

Códigos de prefixo e suas variações têm sido bastante estudados tanto pela comunidade de teoria da informação quanto pela comunidade de ciência da computação. Os códigos de prefixo desempenham papel fundamental nas áreas de compactação e transmissão de dados [20, 36, 38], tendo também aplicações em problemas de busca [3].

Um trabalho pioneiro em códigos de prefixo é devido a Huffman [11]. Neste trabalho, ele considera a construção de códigos de prefixo com mínima redundância. Estes códigos propiciam a melhor taxa de compressão possível sob determinadas condições. A idéia básica é atribuir palavras códigos mais longas aos símbolos menos freqüentes e palavras códigos mais curtas aos símbolos mais freqüentes na mensagem a ser codificada. Huffman conseguiu estabelecer a melhor forma de relacionar o tamanho da palavra código que deve ser associada a um símbolo com sua probabilidade de ocorrência na mensagem a ser codificada.

Nesta tese, apresentamos novos resultados estruturais e algorítmicos sobre a classe dos códigos de prefixo. Explicamos teoricamente as boas taxas de compressão observadas para alguns métodos utilizados na prática. Propomos também algoritmos eficientes para construção de códigos de prefixo ótimos e variantes. Os principais resultados aqui descritos são os seguintes:

1. Um novo algoritmo paralelo para construção de códigos de prefixos ótimos;

2. Uma cota superior para a perda de compressão introduzida pela restrição de comprimento nos códigos de prefixo;

3. Uma cota superior para a perda de otimalidade introduzida pela restrição de altura em árvores alfabéticas de busca;

4. Um algoritmo aproximativo com complexidade $O(n \log n)$ para construção de códigos de prefixo com restrição de comprimento;

5. Um algoritmo aproximativo com complexidade $O(n \log n)$ para construção de árvores alfabéticas de busca com restrição de altura ;

6. Uma nova versão do algoritmo WARM-UP com complexidade fortemente polinomial;

7. Um algoritmo linear para reconhecer códigos de prefixo ótimos com restrição de comprimento;

8. Uma prova afirmativa da conjectura de Vitter sobre o desempenho dos códigos de Huffman dinâmicos construídos pelo algoritmo FGK (Faller, Gallagher e Knuth).

Esta tese foi realizada em 2 anos (Março de 1997 a Julho de 1999). Abaixo listamos artigos derivados desta tese que já foram aceitos para publicação em periódicos e congressos.

- MILIDIÚ, LABER and PESSOA "Bounding the Compression Loss of the FGK Algorithm". Journal of Algorithms 28.

- MILIDIÚ, LABER "Improved Bounds on the Inefficiency of Length Restricted Prefix Codes". Accepted in Algorithmica.

- MILIDIÚ, LABER, "Linear Time Recognition of Length-Restricted Prefix Codes". Proccedings of LATIN 2000, To appear in Lecture notes on Computer Science.

- MILIDIÚ, LABER and PESSOA "A Work Efficient Parallel Algorithm for Constructing Huffman Codes". Proccedings of IEEE Data Compression Conerence 1999.

- LABER, MILIDIÚ, PESSOA "Practical Constructions of L-Restricted Alphabetic Prefix Codes" Proceedings of SPIRE'99.

- MILIDIÚ, PESSOA, LABER. "Efficient implementation of the WARM-UP algorithm for the construction of length-restricted prefix codes". Proceedings of the ALENEX: Lecture Notes in Computer Science 1619.

Além disso, os seguintes artigos estão submetidos a periódicos internacionais.

- MILIDIÚ, LABER, "WARM-UP ALGORITHM: A Lagrangean Construction of Length Restricted Huffman Codes" Submetido ao Siam Journal on Computing

- LABER, MILIDIÚ and PESSOA "Practical Construction of L-restricted Alphabetic Trees" Submetido ao Information Processing Letters;

Nas seções seguintes explicamos com algum detalhe os principais resultados obtidos na tese.

2 Algoritmo de Huffman Paralelo

Um modelo de computação paralela bastante estudado é o modelo PRAM (Parallel Random Access Machine) [12]. Neste modelo, cada processador executa um programa local, e a comunicação entre os processadores é realizada através de uma memória compartilhada. Existem algumas variações do modelo PRAM que dependem do tipo de concorrência permitida nas operações de leitura e escrita em memória compartilhada. O modelo menos restritivo é o modelo CRCW PRAM, onde leitura concorrente e escrita concorrente são permitidas. No modelo CREW PRAM somente leitura concorrente é permitida. O modelo mais restritivo é o modelo EREW PRAM, onde nem leitura nem escrita concorrente são permitidas.

A classe de complexidade NC é definida como a classe dos problemas que admitem um algoritmo que utilize um número polinomial de processadores e que tenha complexidade de tempo polilogaritmico, ou seja, $O(\log^c n)$ para alguma constante $c > 0$.

Alguns algoritmos paralelos foram propostos para construção de códigos de Huffman para o modelo de computação paralela PRAM [4, 19, 13]. O melhor algoritmo pertencente a classe NC é devido a Atallah et al [4]. Este algoritmo é baseado em programação dinâmica. O algoritmo tem complexidade de tempo $O(\log^2 n)$ e utiliza $O(n^2/\log n)$ processadores. Em [19], Larmore e Przytycka reduzem o problema de construir um código de Huffman ao problema de encontrar uma CLWS (Concave Least Weight Subsequence). Além disso, os autores apresentam um algoritmo paralelo para encontrar uma CLWS. O algoritmo tem complexidade de tempo $O(n^{1/2} \log n)$ e requer n processadores.

Nesta tese, propomos e analisamos o algoritmo ES-batched-Huffman que constrói árvores de Huffman no modelo de computação CREW PRAM. O algoritmo tem complexidade de tempo $O(H \log\log(n/H))$ e o seu trabalho é $O(n)$, onde H é o comprimento do maior código gerado. Por trabalho, entenda o total de operações realizadas pelo algoritmo [12]. Como H está no intervalo $[\lceil \log n \rceil, n-1]$, no pior caso o algoritmo tem complexidade de tempo $O(n)$ e no melhor caso $O(\log n \log\log n)$. Provamos também que a complexidade de tempo é $O(\min(n, \log(1/p_1) \log\log n)$, onde p_1 é a menor probabilidade da lista de entrada. Portanto, nosso algoritmo supera o algoritmo de Larmore e Przytcka's [19] quando $\log(1/p_1)$ é $o(\sqrt{n} \log n / \log\log n)$ e o de Atallah et. al. [4] quando $\log(1/p_1)$ é $o(\log^2 n / \log\log n)$. Os resultados desta seção foram publicados em [26].

A idéia básica do algoritmo é aproveitar o fato de que alguns pares de nós podem ser combinados em paralelo durante a execução do algoritmo de Huffman sequencial. A figura 1 mostra um exemplo da execução do algoritmo para a lista de probabilidades $[.04, .06, .06, .06, .08, .12, .14, .20, .24]$. Como exemplo, no passo 1 a folha de probabilidade .04 é combinada com a de probabilidade .06 e duas outras folhas de probabilidade .06 são combinadas. Estas combinações são feitas em paralelo.

3 Códigos de Prefixo com Restrição de Comprimento

3.1 O Problema

Um código de prefixo com restrição de comprimento L é um código de prefixo onde o comprimento de toda palavra código é menor ou igual a L. Por simplicidade utilizamos a sigla CPRC para denotar os códigos de prefixo com restrição de comprimento. Dado um alfabeto $\Sigma = \{a_1, \ldots, a_n\}$, uma lista de probabilidades $[p_1, \ldots, p_n]$ e um inteiro $L \geq$

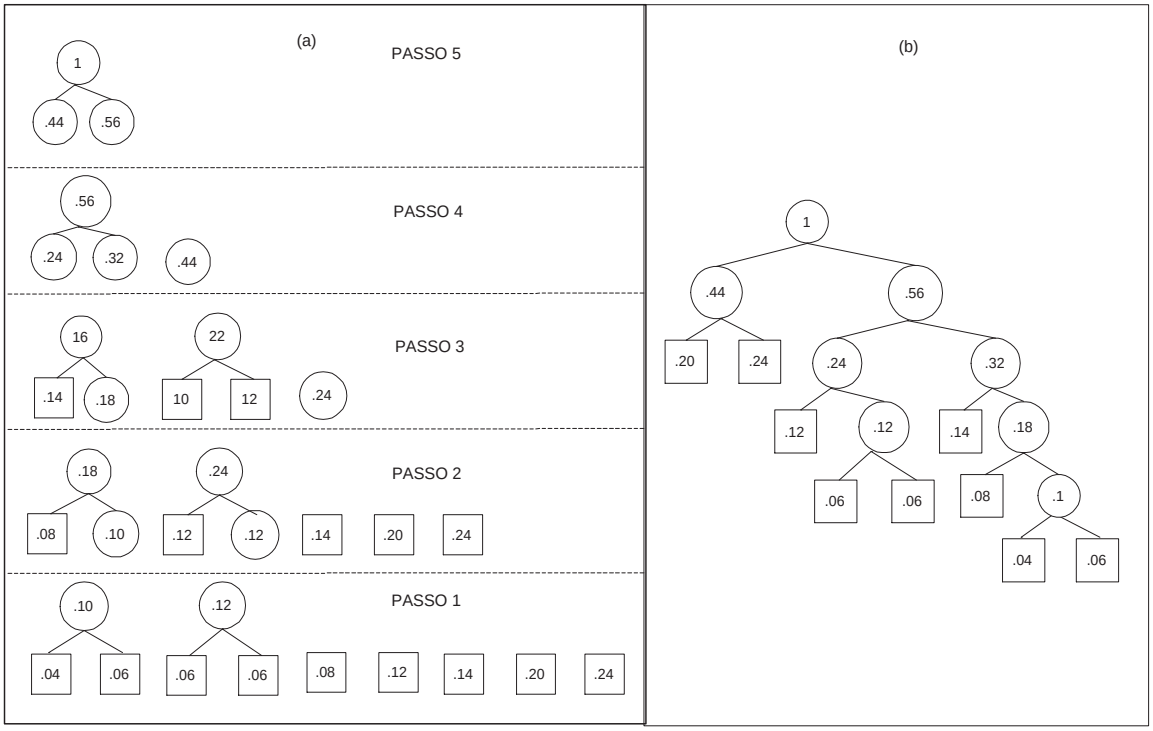

Figura 1: (a) execução do algoritmo paralelo para lista de probabilidades $[.04, .06, .06, .06, .08, .12, .14, .20, .24]$. (b) árvore obtida ao término do algoritmo.

$\lceil \log n \rceil$, um CPRC L-ótimo é um código de prefixo que minimiza o comprimento médio $\sum_{i=1}^{n} p_i l_i$ sujeito à restrição $l_i \leq L$, para $i = 1, \ldots, n$, onde l_i é o comprimento da palavra código associada ao símbolo a_i.

Além do interesse combinatório, a construção de CPRC 's ótimos é interessante em algumas situações práticas. Uma aplicação é evitar uma grande perda de compressão no caso em que as probabilidades de entrada estejam mal estimadas [10]. Outra aplicação é forçar que toda palavra código caiba dentro de uma palavra de computador (16 ou 32 bits). Os decodificadores mais eficientes para códigos de Huffman se utilizam desta propriedade para acelerar consideravelmente a velocidade de decodificação [38, 29].

Entretanto, a restrição de comprimento provoca uma perda de compressão em relação a compressão gerada por um código de prefixo irrestrito (código de Huffman). Duas questões surgem naturalmente.

1. *Os CPRC 's garantem boas taxas de compressão ?*
2. *Existem métodos eficientes para construir CPRC 's ótimos?*

De fato, de que adianta aumentar a velocidade de decodificação se para isso é necessário degradar consideravelmente a qualidade da compressão. Por outro lado, de nada adiantaria que os CPRC's permitissem uma boa compressão se eles não pudessem ser construídos dentro de um limite de tempo razoável.

Nesta tese, abordamos estas duas questões.

3.2 Cotas superiores e um algoritmo aproximado

Sejam A e A_L, respectivamente, o comprimento médio de um código de Huffman para Σ e o comprimento médio de um CPRC L-ótimo para Σ. Além disto, seja E a entropia de Shannon definida por $E = -\sum_{i=1}^{n} p_i \log p_i$. A redundância r_L de um CPRC L-ótimo é definida por $r_L = A_L - E$. Cotas superiores para r_L são dadas em [10, 5]. Capocelli e De Santis [5] provam que $r_L \leq 1 - \log(1 - n2^{1-L})$, para $L > 1 + \log n$. Este resultado limita r_L superiormente por um valor maior que $1 + 2^{\log n + 1 - L}$. Entretanto, esta cota não é capaz de prever a pequena redundância observada em experimentos práticos [10, 31, 29]. De fato, estes experimentos sugerem que a perda de compressão devida a utilização de CPRC's L-ótimos em vez de códigos de Huffman é quase desprezível.

Definimos ϵ como $A_L - A$. Esta diferença mede o número extra de bits por símbolo resultante da utilização de CPRC 's L-ótimos em vez de códigos de Huffman. O valor de ϵ não pode ser menor que 0, já que o comprimento médio de um código de Huffman não pode ser maior que o comprimento médio de um CPRC L-ótimo. Dependendo da lista de probabilidades e do valor de L, ϵ assume diferentes valores. Nesta tese mostramos que $\epsilon < \lceil \log n \rceil - 1$, quando $L = \lceil \log n \rceil$. Para $L \geq \lceil \log n \rceil + 1$, mostramos que

$$\epsilon \leq 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 1}, \tag{1}$$

onde ψ é a razão áurea 1.618. Uma consequência imediata destas cotas é que $\epsilon \leq 0.01$ para $L \geq \lceil \log n \rceil + 11$. Este resultado implica que a perda de compressão devida ao uso de CPRC 's L-ótimos é praticamente desprezível para $L \geq \lceil \log n \rceil + 11$. Como $\lceil \log n \rceil + 11 < 32$ para alfabetos com menos de 2.000.000 de símbolos, podemos fixar L em 32 (tamanho da palavra de computador) com garantia de perda de compressão inferior a 1%.

A cota superior para ϵ é derivada da análise de aproximação do BRCI, um novo algoritmo para construção de códigos de prefixo com restrição de comprimento. O BRCI é um algoritmo aproximado, no sentido que constrói códigos subótimos. Sua complexidade de tempo é $O(n \log n)$ e sua complexidade de espaço é $O(n)$. Se as probabilidades estão ordenadas o algoritmo tem complexidade de tempo linear. Um pseudo-código para o algoritmo é apresentado abaixo e um exemplo aparece na figura 2. O algoritmo BRCI e sua análise aparecem em [24]

O Algoritmo BRCI	
Passo 0:	Construa uma árvore de Huffman T para a lista de probabilidades dada.
Passo 1:	Remova todas subárvores de T que tem como raízes os nós situados no nível L. Seja T^1 a árvore remanescente.
Passo 2:	Crie uma árvore completa T^2 com as folhas das subárvores removidas no passo 1. Seja $h(T^2)$ a altura da árvore T^2.
Passo 3:	Seja y_p o nó de T^1 com menor probabilidade situado no nível $L - h(T^2) - 1$ em T^1 e seja T_p a subárvore enraizada em y_p. Crie um novo nó y^*. Se y_p não é raiz da árvore T^1, então insira o nó y^* entre y_p e o pai de y_p. Se y_p é raiz de T^1, então faça y^* ser pai de y_p. Seja T^3 a árvore remanescente.
Passo 4:	Faça a árvore T^2 ser filha à direta do nó y^* em T^3. Seja T_R a árvore obtida.

Com base no algoritmo BRCI e em algumas idéias que aparecem em [37], obtemos um algoritmo aproximado com complexidade $O(n \log n)$ para construir árvores alfabéticas de

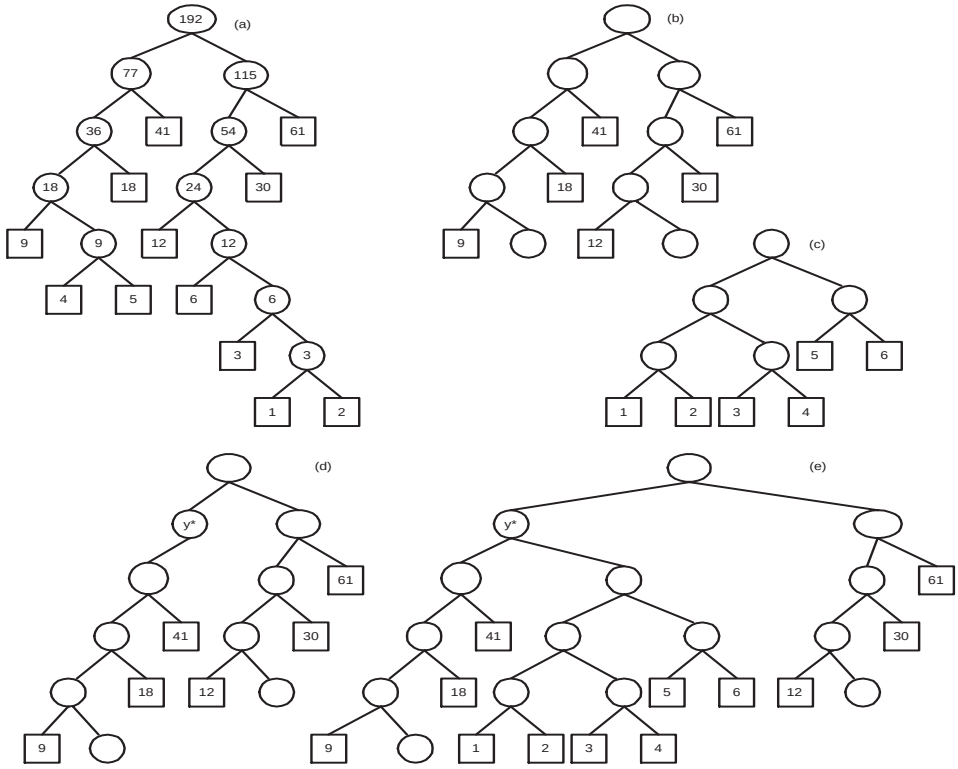

Figura 2: Exemplo do algoritmo BRCI para uma lista de probabilidades $[1/192, 2/192, 3/192, 4/192, 5/192, 6/192, 9/192, 18/192, 30/192, 41/192, 61/192]$ e $L = 5$. (a) Árvore de Huffman T. (b) Árvore T_1 obtida pela remoção dos nós em nível maior ou igual a 5 na árvore T. (c) árvore completa T_2 para os nós removidos em T. (d) Árvore T_3 obtida a partir da inserção de um nó artificial y^* na árvore T_1. (e) árvore com restrição de altura 5 obtida a partir de T_2, fazendo com que a árvore T_2 se torne filho a direita do nó artificial y^*.

busca com restrição de altura [9, 18]. A partir da análise de aproximação deste algoritmo, conseguimos uma cota superior para a perda de otimalidade introduzida pela restrição de altura. Conforme o survey de Abrahams [1], não se conhecia cota superior para esta perda. Esta cota implica que é sempre possível construir estratégias de busca que realizem na média poucas comparações e que ao mesmo tempo não permitam que uma busca individual seja muito cara, ou seja, gaste mais que L comparações. Estes resultados aparecem em [16].

3.3 Uma Versão Fortemente Polinomial para um Algoritmo Lagrangeano

Um outro resultado desta tese é uma versão fortemente polinomial para o algoritmo WARM-UP proposto em [23, 15]. O WARM-UP é um algoritmo bastante simples e rápido para geração de códigos de prefixo com restrição de comprimento. Para benchmarks da literatura, como o Calgary Corpus e a coleção Reuters, o algoritmo consegue obter um código ótimo com restrição de comprimento, construindo duas ou três árvores de Huffman [27]. Apesar do WARM-UP ser um algoritmo aproximado, ele pode ser convertido em um

algoritmo ótimo com um custo adicional de $O(n \log n)$ [15].

A estratégia do WARM-UP é buscar um valor racional x^* tal que a árvore de Huffman para a lista de probabilidades $(\max\{x^*, p_1\}, \ldots, \max\{x^*, p_n\})$ tenha altura L. Mais especificamente, escolhe-se um valor x e constrói-se uma árvore de Huffman para a lista de probabilidades modificadas $(\max\{x, p_1\}, \ldots, \max\{x, p_n\})$. Se a árvore obtida tiver altura maior que L, o algoritmo aumenta o valor de x e constrói uma árvore novamente. Se a árvore obtida tiver altura menor que L, o algoritmo diminui o valor de x e constrói uma árvore novamente. Finalmente, se a árvore tiver altura L, o algoritmo retorna o código de prefixo induzido por esta árvore.

Utilizando resultados clássicos de dualidade Lagrangeana [28] e alguns lemas combinatórios adicionais, em [15] provamos que a árvore de Huffman obtida a partir da lista de probabilidades modificada pelo parâmetro x^* resolve ou aproxima bastante o problema com restrição de comprimento. Neste mesmo trabalho, provamos uma relação de monotonicidade que mostra que o aumento deste parâmetro x implica em uma redução (não aumento) da altura da árvore, justificando portanto uma busca binária para encontrar x^*.

A busca binária para encontrar o valor x^* faz com que a complexidade de tempo do WARM-UP não seja fortemente polinomial. De fato, sua complexidade depende do inverso do logaritmo da menor probabilidade $-\log p_{min}$. Para obter uma versão fortemente polinomial para o WARM-UP combinamos um esquema de busca paramétrica introduzido em [21] com uma versão sequencial do algoritmo ES-Batched-Huffman proposto nesta tese. Esta versão tem complexidade de tempo $O(nL\log(n/L))$ e utiliza espaço $O(n)$. Este resultado ilustra mais uma vez, o fato nada intuitivo, de que o design de algoritmos paralelos eficientes pode conduzir a um ganho de desempenho em algoritmos sequenciais. Em nosso caso, conseguimos melhorar a complexidade de tempo do algoritmo WARM-UP utilizando uma versão sequencial do algoritmo paralelo ES-Batched-Huffman.

3.4 Reconhecimento de Códigos de Prefixo Ótimos em Tempo Linear

Apesar do esforço de alguns pesquisadores [10, 8, 33, 17, 2, 30, 31, 32], continua em aberto se existe algum algoritmo ótimo com complexidade $O(n \log n)$ para construir códigos de prefixo com restrição de comprimento.

Na tese, apesar de não conseguirmos provar este resultado, mostramos que os CPRC's ótimos podem ser reconhecidos em tempo linear se a lista de probabilidades de entrada estiver ordenada, e como consequência, em $O(n \log n)$ para uma lista de probabilidades não ordenada. A prova da corretude do algoritmo é extremamente técnica, sendo composta de uma lista extensa de argumentos combinatórios que tomam como base uma representação especial das árvores binárias denominada de nodesets [17].

Para os inteiros positivos i e h, definimos um bloco como um par ordenado (i, h), onde i é o índice do bloco e h é o nível do bloco. Um conjunto de blocos é denominado nodeset. Definimos o nodeset $R(n, L)$ por

$$R(n, L) = \{(i, h) | 1 \leq i \leq n, 1 \leq h \leq L\}$$

Seja T uma árvore binária de n folhas, cujos níveis das folhas são $l_1 \geq \ldots \geq l_n$. O nodeset $N(T)$ associado a T é definido por $N(T) = \{(i, h) | 1 \leq h \leq l_i, 1 \leq i \leq n\}$

A figura 3 mostra $R(8,5)$. Os blocos dentro do polígono são os blocos que constituem $N(T)$, onde T é a árvore binária com folhas em níveis $5,5,5,5,3,3,3,1$.

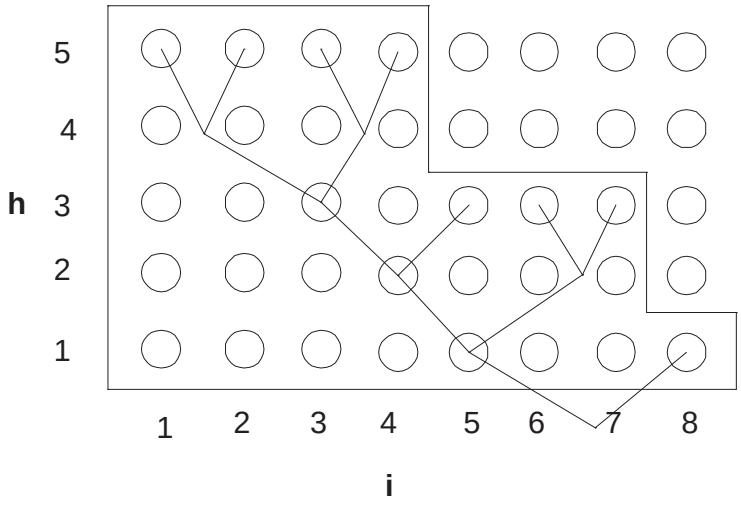

Figura 3: O nodeset $R(8,5)$ e o nodeset $N(T)$ associado a árvore binária T com folhas em níveis $5,5,5,5,3,3,3,1$

Associamos uma função de peso e uma função de largura aos blocos do nodeset. Dado um bloco (i,h), definimos $largura(i,h) = 2^{-h}$ e $peso(i,h) = p_i$. A largura e o peso de um nodeset, são definidos como a soma dos blocos que constituem o nodeset. Em [17], Larmore e Hirschberg reduziram o problema de encontrar um código de prefixo ótimo com restrição de comprimento L para uma lista de probabilidades $p_1 \leq \cdots \leq p_n$ ao problema de encontrar o nodeset com largura $n-1$ e peso mínimo contido em $R(n,L)$.

Para encontrar o nodeset com largura $n-1$ e peso mínimo contido em $R(n,L)$, Larmore e Hirschberg propõem o algoritmo Package-Merge [17]. Este resolve o seguinte problema: dado um nodeset R e uma largura d, encontre o nodeset $X \subset R$, com largura d e peso mínimo. O Package-Merge utiliza uma abordagem gulosa e tem complexidade de tempo $O(|R|)$, onde $|R|$ é o número de blocos do nodeset R. O algoritmo de Reconhecimento proposto nesta tese utiliza o Package-Merge como procedimento auxiliar.

Para determinar em tempo linear se um determinado código de prefixo é ótimo ou não, consideramos o nodeset N associado a árvore que representa tal código. A partir de então definimos alguns conjuntos de blocos e provamos que o reconhecimento pode ser feito aplicando o algoritmo Package-Merge a união destes conjuntos. A complexidade linear decorre do fato de que a união destes conjuntos contém no máximo $8n+2L = O(n)$ blocos. A intuição é que se o nodeset N não é ótimo, sempre podemos diminuir seu peso incluindo blocos que estejam próximos a linha poligonal que define o nodeset e retirando blocos que também estejam próximos a esta mesma linha poligonal. Como exemplo, a linha poligonal na figura 4define o nodeset associado a uma árvore binária com folhas em níveis $10,10,10,10,9,9,7,6,6,6,4,3,2,1$. Se este nodeset não é ótimo, qualquer que seja a lista de probabilidades de entrada, podemos mostrar que é possível diminuir seu peso (mantendo sua largura) através da substituição de alguns blocos negros fora do polígono por alguns blocos negros situados dentro do polígono.

A possibilidade de reconhecer CPRC's L-ótimos em tempo linear ilustra o poder da representação por nodesets. Os nodesets tornam mais fácil o entendimento de várias

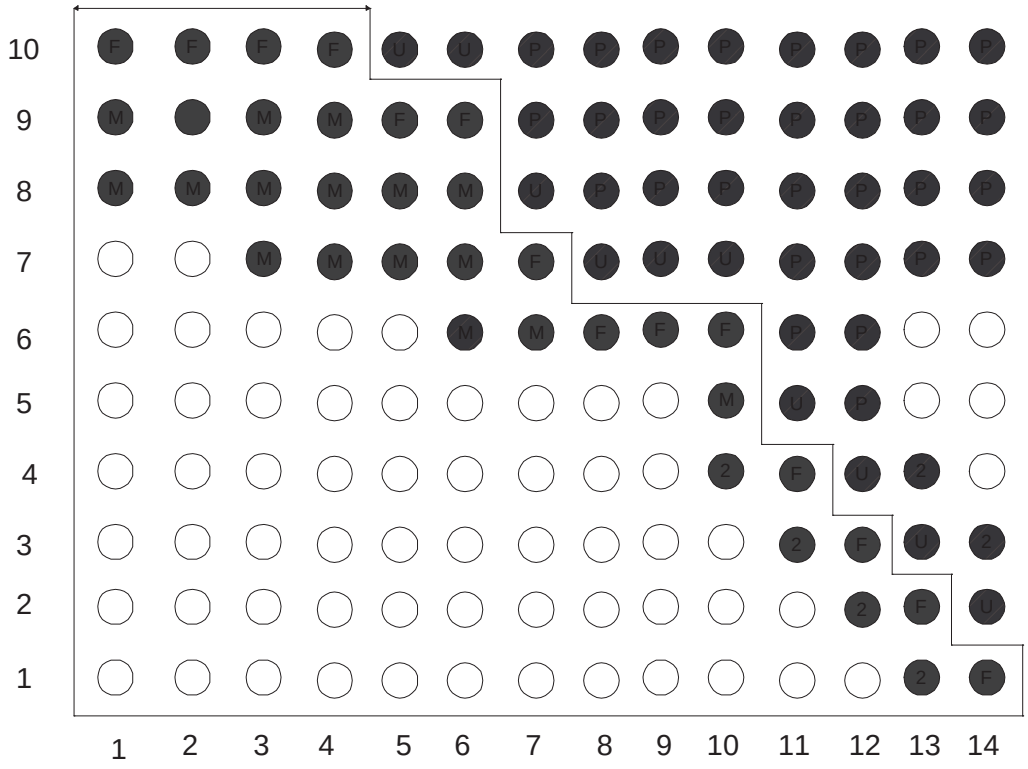

Figura 4: Os nodesets utilizados pelo algoritmo de reconhecimento

propriedades dos códigos de prefixo e das árvores de codificação. Algumas propriedades que são difíceis de analisar em uma árvore, como algumas modificações locais, podem ser tratadas de modo mais natural através da representação por nodesets. Resultados desta subseção aparecem em [25]. Uma versão estendida será submetida a algum periódico internacional.

4 Códigos de Huffman Dinâmicos

Em várias aplicações de compressão, as probabilidades dos símbolos do alfabeto de entrada são estimadas contando o número de ocorrências de cada símbolo na mensagem a ser comprimida. No entanto, para fins de transmissão de dados este esquema não é razoável, já que o transmissor não pode esperar que toda mensagem esteja disponível até começar a transmissão. Nestas situações, os códigos de Huffman dinâmicos são uma alternativa a ser considerada. A compressão dinâmica de Huffman determina um mapeamento do alfabeto de entrada nas palavras código baseando-se na estimativa corrente dos pesos dos símbolos que compõem o alfabeto de entrada. O código vai se modificando de modo a se manter ótimo para a estimativa corrente.

Como exemplo, vamos considerar que a mensagem **acacbcabddfabafecaa** já foi transmitida. A árvore de Huffman para esta mensagem é mostrada na figura 5.(a). Sendo **b** o próximo símbolo da mensagem, o transmissor transmite a palavra código 010 associada a **b**, e em seguida atualiza a árvore de Huffman, já que o peso de **b** mudou de 3 para 4. A nova árvore de Huffman é mostrada na figura 5.(b). Ao receber a palavra código

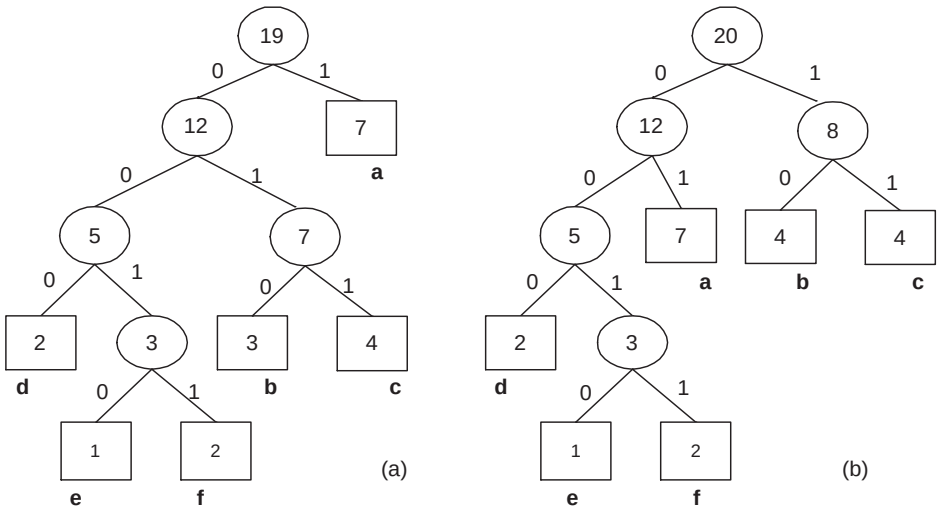

Figura 5: (a) mostra a árvore de Huffman antes da chegada do símbolo **b**. (b) mostra a nova árvore após a transmissão de **b**.

010, o receptor obtém o símbolo **b**, e atualiza a árvore de Huffman de forma idêntica ao transmissor.

Os códigos de Huffman dinâmicos foram desenvolvidos independentemente por Faller e Gallager [6, 7]. Em [14], D. E. Knuth propôs uma implementação eficiente para gerar tais códigos. O algoritmo resultante dos esforços destes autores, é conhecido como algoritmo FGK. Outra versão para códigos de Huffman dinâmicos é o algoritmo Λ descrito por Vitter [34, 35]. Ambos algoritmos gastam um tempo $O(l)$ para atualizar a árvore de Huffman, onde l é o comprimento da palavra código atribuída ao símbolo transmitido.

Seja um alfabeto $\Sigma = \{a_1, \ldots, a_n\}$ e seja M uma mensagem qualquer de comprimento t. Em [34], Vitter provou que o algoritmo Λ utiliza no máximo $S_t + t - 2n$ bits para codificar M, onde S_t é o número de bits utilizados pelo algoritmo de Huffman estático para codificar M. No mesmo artigo Vitter também provou que o número total de bits D_t utilizados pelo algoritmo FGK para codificar M é cotado superiormente por $2S_t+t-4n+2$. Esta última cota garante que o número médio de bits por caracter gasto pelo FGK não é muito maior do que o dobro do gasto pelo código de Huffman estático na codificação de uma mesma mensagem. No entanto, Vitter observa que o desempenho do FGK é bem melhor do que o desempenho previsto por sua cota superior, o que o leva a conjecturar que $D_t = S_t + O(t)$ [34]. A validade conjectura implica que a diferença entre o número médio de bits por caracter utilizado pelo FGK e o utilizado pelo algoritmo de Huffman estático, para codificar a mesma mensagem, é limitada por uma constante.

Nesta tese, provamos que a conjectura é verdadeira, mostrando que $D_t \leq S_t + 2t - 2k - \lceil \log \min(k+1, n) \rceil$, onde k é o número de símbolos distintos na mensagem. Este resultado implica que $(D_t - S_t)/t < 2$, o que mostra que o número médio de bits por caracter utilizado pelo FGK é no máximo duas unidades maior do que o utilizado pelo algoritmo de Huffman estático. Também apresentamos um exemplo onde $D_t = S_t + 2t - 2k - 3\lfloor (t-k)/k \rfloor - \lceil \log(k+1) \rceil$. Este exemplo mostra que nossa cota é assintoticamente justa. Estes resultados explicam o bom desempenho do algoritmo FGK observado por alguns autores através de experimentos práticos [14, 34].

Explicamos em linhas gerais as idéias envolvidas na prova da conjectura. Em [34],

Vitter mostra que o número de bits adicionais transmitidos pelo FGK em relação ao código de Hufffman estático é cotado superiormente por $\sum_{i=1}^{t} d_i$, onde d_i é a diferença entre o nível da folha correspondente ao i-ésimo símbolo transmitido, antes e depois da atualização da árvore de Huffman. Como exemplo, no caso da figura 5, a folha correspondente ao símbolo b tem nível 3 antes da atualização e nível 2 depois da atualização. Portanto, $d_{20} = 3 - 2 = 1$. Entretanto, Vitter não consegue obter uma cota superior razoável para $\sum_{i=1}^{t} d_i$. Utilizando uma nova técnica de decomposição de árvores e argumentos de análise amortizada que tomam como função potencial a altura da árvore de Huffman, conseguimos mostrar que $\sum_{i=1}^{t} d_i \leq 2t$, estabelecendo portanto a conjectura. Os resutados desta seção aparecem em [22].

Referências

[1] J. Abrahams. Code and parse trees for lossless source encoding. In *Proceedings of Compression and Complexity of Sequences 1997*, pages 145–171. IEEE Computer Society, 1997.

[2] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum-weight k-link path in graphs with the concave monge property and applications. *GEOMETRY: Discrete & Computational Geometry*, 12:263–280, 1994.

[3] R. Ahlswede and R. Wegener. *Search Problems*. John Wiley and Sons, New York, New York, 1987.

[4] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S.-H. Teng. Constructing trees in parallel. In A.-S. ACM-SIGARCH, editor, *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, page 421, Santa Fe, NM, June 1989. ACM Press.

[5] R. M. Capocelli and A. D. Santis. On the redundancy of optimal codes with limited word length. *IEEE Transactions on Information Theory*, 38:439–445, 1992.

[6] N. Faller. An adaptive system for data compression. In *Record of the 7th Asilomar Conferences on Circuits, Systems and Computers*, pages 593–597, 1973.

[7] R. Gallager. Variations on a theme of huffman. *IEEE Transaction on Information Theory*, 24(6):668–674, November 1978.

[8] M. R. Garey. Optimal binary search trees with restricted maximal depth. *Siam Journal on Computing*, 3(2):101–110, June 1974.

[9] A. M. Garsia and M. L. Wachs. A new algorithm for minimum cost binary trees. *SIAM Journal on Computing*, 6(4):622–642, Dec. 1977.

[10] E. N. Gilbert. Codes based on innacurate source probabilities. *IEEE Transactions on Information Theory*, 17:304–314, 1971.

[11] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. Inst. Radio Eng.*, pages 1098–1101, Sept. 1952. Published as Proc. Inst. Radio Eng., volume 40, number 9.

[12] J. Jaja. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, 1992.

[13] Kirkpatrick and Przytycka. Parallel construction of binary trees with near optimal weighted path length. *Algorithmica*, 15, 1996.

[14] D. E. Knuth. Dynamic Huffman coding. *Journal of Algorithms*, 6(2):163–180, June 1985.

[15] E. S. Laber. Um algoritmo eficiente para construção de códigos de prefixo com restrição de comprimento. Master's thesis, PUC-RJ, 1997.

[16] E. S. Laber, R. L. Milidiú, and A. A. Pessoa. Practical constructions of l-restricted alphabetic prefix codes. In A. L. Oliveira, editor, *Proceedings of SPIRE'99*, pages 115–119, Cancun, Mexico, September 1999.

[17] L. L. Larmore and D. S. Hirschberg. A fast algorithm for optimal length-limited Huffman codes. *Journal of the ACM*, 37(3):464–473, July 1990.

[18] L. L. Larmore and T. M. Przytycka. A fast algorithm for optimum height-limited alphabetic binary trees. *SIAM Journal on Computing*, 23(6):1283–1312, December 1994.

[19] L. L. Larmore and T. M. Przytycka. Constructing Huffman trees in parallel. *SIAM Journal on Computing*, 24(6):1163–1169, Dec. 1995.

[20] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, Sept. 1987.

[21] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, Oct. 1983.

[22] Milidiu, Laber, and Pessoa. Bounding the compression loss of the FGK algorithm. *Journal of Algorithms*, 28:195–211, August.

[23] R. L. Milidiú and E. S. Laber. Warm-up algorithm: A lagrangean construction of length restricted huffman codes. Technical Report 15, Departamento de Informática, PUC-RJ, Rio de Janeiro, Brasil, January 1996.

[24] R. L. Milidiú and E. S. Laber. Improved bounds on the inefficiency of length restricted codes. *To appear in Algorithmica*, 2000.

[25] R. L. Milidiú and E. S. Laber. Linear time recognition of optimal l-restricted prefix codes. In G. Gonnet, D. Panario, and A. Viola, editors, *Proceedings of LATIN 2000*, Lecture Notes in Computer Science, page To appear, Punta Del Leste, Uruguay, April 2000.

[26] R. L. Milidiú, E. S. Laber, and A. A. Pessoa. A work efficient parallel algorithm for constructing huffman codes. In J. A. Storer and M. Cohn, editors, *Proceedings of the IEEE Data Compression Conference*, pages 277–286, Snowbird, Utah, USA, March 1999.

[27] R. L. Milidiu, A. A. Pessoa, and E. S. Laber. Efficient implementation of the WARM-UP algorithm for the construction of length-restricted prefix codes. In *Proceedings of the ALENEX*, volume 1619 of *Lecture Notes in Computer Science*, Baltimore, Maryland, USA, January 1999. Springer.

[28] M. Minoux. *Mathematical Programming, Theory and Algorithms.* John Wiley & Sons, 1986.

[29] A. Moffat and A. Turpin. Efficient construction of minimum-redundancy codes for large alphabets. *IEEE Transactions on Information Theory*, 44(4):1650–1657, July 1998.

[30] B. Schieber. Computing a minimum-weight k-link path in graphs with the concave Monge property. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 405–411, San Francisco, California, 22–24 Jan. 1995.

[31] A. Turpin and A. Moffat. Practical length-limited coding for large alphabets. *The Computer Journal*, 38(5):339–347, 1995.

[32] A. Turpin and A. Moffat. Efficient implementation of the package-merge paradigm for generating length-limited codes. In M. E. Houle and P. Eades, editors, *Proceedings of Conference on Computing: The Australian Theory Symposium*, pages 187–195, Townsville, 29–30 Jan. 1996. Australian Computer Science Communications.

[33] D. C. van Voorhis. Constructing codes with bounded codeword lengths. *IEEE Transactions on Information Theory*, 20:288–290, 1974.

[34] J. S. Vitter. Design and analysis of dynamic Huffman codes. *Journal of the ACM*, 34(4):825–845, Oct. 1987.

[35] J. S. Vitter. Dynamic huffman coding. *ACM Trans. Math. Softw.*, 15(2):158–167, June 1989.

[36] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, NY, USA, 1994. The software for full-text indexing described in this book, and errata for the book, are available for anonymous ftp from path=munnari.oz.au= in the directory path=/pub/mg=.

[37] R. W. Yeung. Alphabetic codes revisited. *IEEE Transactions on Information Theory*, 37(3):564–572, May 1991.

[38] J. Zobel and A. Moffat. Adding compression to a full-text retrieval system. *Software—Practice and Experience*, 25(8):891–903, Aug. 1995.