The WARM-UP Algorithm: A Lagrangean Construction of Length Restricted Huffman Codes

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

PUC-RioInf.MCC15/96 September, 1996

Abstract: Given an alphabet $\{a_1, \ldots, a_n\}$ with corresponding set of weights $\{w_1, \ldots, w_n\}$, and a number $L \geq \lceil \log n \rceil$, we introduce an $O(n \log n + n \log \overline{w})$ algorithm for constructing a suboptimal prefix code with restricted maximal length L, where $\overline{w}$ is the highest presented weight. The number of additional bits per symbol generated by our code is not greater than $1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2}$ when $L > \lceil \log n \rceil + 1$, where ψ is the golden ratio 1.618.

An important feature of the proposed algorithm is its implementation simplicity. The algorithm is basically a selected sequence of Huffman trees construction for modified weights.

Keywords: Prefix codes, Huffman Trees, Lagragean Duality

Resumo: Dado um alfabeto $\{a_1, \ldots, a_n\}$ com pesos correspondentes $\{w_1, \ldots, w_n\}$ e um número $L \geq \lceil \log n \rceil$, nós apresentamoso um algoritmo de de complexidade $O(n \log n + n \log \overline{w})$para construit códigos de prefixo subótimos com restrição de comprimento L, onde $\overline{w}$ é o maior peso do dado conjunto. O número de bits adicionais por símbolo gerados por nosso código não ultrapassa $1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2}$ para $L > \lceil \log n \rceil + 1$, onde ψ é a razão aúrea 1.618.

Um importante aspecto do algoritmo proposto é a simplicidade de sua implementção. O algoritmo ćonstitui basicamente de uma sucessiva construção e árvores de Huffman para um conjunto modificado de pesos.

Palavras-Chave: Códigos de Prefixo, 'Arvores de Huffman, Relaxação Lagrangeana.

1 Introduction

An important problem in the field of Coding and Information Theory is the Variable Length Code Problem [Huf52]. A particular case of relevant interest is the Binary Prefix Code Problem. Given an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ and a corresponding set of weights $\{w_1, \ldots, w_n\}$, the problem is to find a prefix code for Σ that minimizes the *weighted length* of a code string, defined to be $\sum_{i=1}^{n} w_i l_i$, where l_i is the length of the code assigned to a_i. This problem is equivalent to the following problem: given a set of weights $\{w_1, \ldots, w_n\}$, find a tree T that minimizes the *weighted path length* $\sum_{i=1}^{n} w_i l_i$, where l_i is height of the ith leaf of T. T must be chosen among all full binary trees[1] with n leaves. The equivalence is due to the fact that every binary prefix code can be represented by a full binary tree [Eve79]. If the set of weights is sorted, this problem can be solved in $O(n)$ by one of the efficient implementations of Huffman's Algorithm [Huf52, Lee76, Ktj95]. Any tree constructed by Huffman's Algorithm is called a Huffman tree.

In this paper we consider the binary prefix code problem with restricted maximal length, that is, for a fixed $L \geq \lceil \log n \rceil$, we must minimize $\sum_{i=1}^{n} w_i l_i$ constrained to $l_i \leq L$ for $i = 1, \ldots, n$. Gilbert [Gil71] recommends to formulate this problem when the weights w_i are inaccurately known. Choueka, Klein and Perl [Cho85] suggest the use of length restricted codes to reduce the external path length $\sum_{i=1}^{n} l_i$. The objective is to allow space efficient decoding of optimal prefix codes without bit-manipulation. Zobel and Moffat [Zob95] describe the use of word-based Huffman codes for compression of large textual databases. The application allows the maximum of 32 bits for each codeword. For the cases that exceed this limitation, it is recommended to use codes with length restriction.

Some methods can be found in the literature to solve the Binary Prefix Code Problem with Restricted Maximal Length [HuT72, Voo74, Fra93]. The first polynomial algorithm is due to Garey [Gar74]. The algorithm is based on dynamic programming and it has an $O(n^2 L)$ complexity for time and space. Larmore and Hirschberg [Lar90] presented the Package-Merge algorithm. This is an $O(nL)$ algorithm that requires linear space. The authors reduce the original problem to a Coin's Collector Problem, using a nodeset representation of a binary tree. Turpin and Moffat [Tur95] discuss some aspects about the implementation of the Package-Merge algorithm. Aggarwal *et al.* [Agg94] have used Megiddo's parametric search paradigm [Meg83] to obtain

[1] In a full binary tree each internal node has exactly two sons.

an $O(n\sqrt{L \log n} + n \log n)$ time algorithm. Currently, the fastest strongly polynomial time algorithm for the problem is due to Schieber [Scb95]. This algorithm also utilizes a parametric search. However, it runs in $O(n2^{o(\sqrt{\log L \log\log n})})$ time and requires $O(n)$ space. These last two algorithms are based on a reduction of the Binary Prefix Code Problem to the Concave Least Weight Subsequence Problem [Lar95].

In this paper we present an approximative algorithm that constructs length restricted prefix codes. The algorithm is called WARM-UP and is based on lagrangean relaxation [Ahu93], a powerful technique used to solve some important problems in *Combinatorial Optimization*. The worst case complexity of the algorithm is given by $O(n \log n + n \log \overline{w})$, where n is the number of weights and $\overline{w}$ is the highest weight. The algorithm requires $O(n)$ space and can be implemented in-place [Mil97b] by using the Huffman procedure presented in [Ktj95].

The first parametric search proposed in [Agg94] is similar to the WARM-UP approach introduced here. Nevertheless, from a lagrangean point of view, the former approach uses only one lagrange multiplier, whereas the later uses a varying number of nonzero lagrange multipliers throughout the WARM-UP process. Furthermore, Aggarwal's algorithm solves a sequence of Concave Least Weight Subsequence Problems, while our algorithm constructs a sequence of Huffman trees.

Due to the simplicity of Huffman's procedure, the WARM-UP ALGORITHM has a small constant factor in its $O(n \log n + n \log \overline{w})$ time complexity. Furthermore, the Warm-up can take advantage of improvements in Huffman's procedure, as the recently one proposed by Moffat and Turpin [Mof97].

The paper is organized as follows. In section 2, we give an integer programming formulation to the problem, and develop the theoretical background needed to justify our algorithm. In section 3, we address the special case of constructing a Huffman tree with height exactly equal to L, given that the Huffman tree with maximum height has height greater than L, and the Huffman tree with minimum height has height smaller than L. In section 4, we introduce the WARM-UP ALGORITHM. A detailed analysis of the approximation and the complexity of the algorithm is presented. Finally, in section 5 we present some conclusions and comments on the implementation aspects of the algorithm and other applications of this approach.

2 The Lagrangean Approach

Our approach is based on an integer programming formulation to the problem [Fra93]. Given a sequence of n integer weights $w_1 \leq \ldots \leq w_n$ and a length $L \geq \lceil \log n \rceil$ the problem is to find a sequence of n integers $l_1, \ldots, l_n$ that minimizes

$$\sum_{i=1}^{n} w_i.l_i$$

constrained to

$$\begin{cases} l_i \leq L \\ \sum_{i=1}^{n} 2^{-l_i} = 1 \qquad\qquad i = 1, \ldots, n \\ l_i \in N \end{cases} \tag{1}$$

We call this problem P_L. The restriction $\sum_{i=1}^{n} 2^{-l_i} = 1$ is satisfied if and only if $l_1, \ldots, l_n$ are the heights of the leaves of a full binary tree with n leaves [Eve79]. In mathematical programming terms we say that P_L is a primal problem [Min93].

Since $w_1 \leq \ldots \leq w_n$, then there is a solution where $l_1 \geq \ldots \geq l_n$. Throughout this work we use this fact. For convenience, we use S_n to denote the points with integer coordinates that satisfy $\sum_{i=1}^{n} 2^{-l_i} = 1$ and we use $h(T)$ to denote the height of a tree T.

2.1 The Lagrangean Relaxation

Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$, with $\lambda_i \geq 0$ for $i = 1, \ldots, n$. The Lagrangean function associated to problem P_L is given by

$$L(\boldsymbol{l}, \boldsymbol{\lambda}) = \sum_{i=1}^{n} w_i.l_i + \sum_{i=1}^{n} \lambda_i.(l_i - L) \tag{2}$$

The values λ_i are called Lagrangean multipliers. A point $(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangean function $L(\boldsymbol{l}, \boldsymbol{\lambda})$ if and only if

(a) $L(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}}) \leq L(\boldsymbol{l}, \overline{\boldsymbol{\lambda}})$ for all $\boldsymbol{l} \in S_n$
(b) $\overline{l_i} \leq L$ for $i = 1, \ldots, n$
(c) $\overline{\boldsymbol{\lambda}}_i.(\overline{l_i} - L) = 0$ for $i = 1, \ldots, n$

It is a well known result [Min93] that if $(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangean function $L(\boldsymbol{l}, \boldsymbol{\lambda})$, then $\overline{\boldsymbol{l}}$ is a global optimum to the primal problem, in this case to problem P_L.

Now, we can establish a sufficient condition for the sequence of integers $l_i, \ldots, l_n$ to be a solution to problem P_L, for the given set of weights $w_1, \ldots, w_n$.

Theorem 1 *Let $\overline{\boldsymbol{\lambda}} = (\overline{\lambda_1}, \ldots, \overline{\lambda_n})$, with $\overline{\lambda_i} \geq 0$ for $i = 1, \ldots, n$. If the vector $\overline{\boldsymbol{l}} = (\overline{l_1}, \ldots, \overline{l_n})$ corresponds to the heights of the leaves of a Huffman tree for the modified set of weights given by $w_1 + \overline{\lambda_1}, \ldots, w_n + \overline{\lambda_n}$, with*

$$\begin{cases} \overline{l}_i \leq L \text{ and} \\ \overline{l}_i = L \text{ if } \overline{\lambda_i} \neq 0, \end{cases}$$

for $i = 1, \ldots, n$, then $\overline{\boldsymbol{l}}$ solves problem P_L.

We show that $(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangean function $L(\boldsymbol{l}, \boldsymbol{\lambda})$ by proving conditions $(a) - (c)$.

(a): Since $\overline{\boldsymbol{l}}$ corresponds to the heights of the leaves of a Huffman tree for the weights $w_1 + \overline{\lambda_i}, \ldots, w_n + \overline{\lambda_n}$, it follows that

$$\sum_{i=1}^{n}(w_i + \overline{\lambda}_i).\overline{l}_i \leq \sum_{i=1}^{n}(w_i + \overline{\lambda}_i).l_i \quad \text{for all } \boldsymbol{l} \in S_n$$

Subtracting the expression $\sum_{i=1}^{n} \overline{\lambda}_i.L$ from both sides of the inequality, we obtain that

$$\sum_{i=1}^{n} w_i.\overline{l}_i + \sum_{i=1}^{n} \overline{\lambda}_i(\overline{l}_i - L) \leq \sum_{i=1}^{n} w_i.l_i + \sum_{i=1}^{n} \overline{\lambda}_i(l_i - L),$$

and therefore

$$L(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}}) \leq L(\boldsymbol{l}, \overline{\boldsymbol{\lambda}}) \text{ for all } \boldsymbol{l} \in S_n.$$

Conditions (b) and (c) follow immediately from the theorem hypothesis. This result shows that $\overline{\boldsymbol{l}}$ is a global optimum to problem P_L.

2.2 Finding the Multipliers

Theorem 1 suggests a simple way to solve problem P_L. Find multipliers $\lambda_1, \ldots, \lambda_n$ such that the Huffman tree for the modified weights $w_1 + \lambda_1, \ldots, w_n + \lambda_n$ have height equal to L, and all the leaves associated to the effectively modified weights $w_i + \lambda_i$, with $\lambda_i > 0$, are arranged at height L. This Huffman tree is an optimal prefix code tree with height restriction L.

As an example, let us consider the set of weights $W = \{1, 1, 2, 3, 5, 8, 13\}$. Figure 1.(a) shows a Huffman tree for this set. Suppose we add 0.5 to the two lowest weights of the given set. The tree in figure 1.(b) is a Huffman tree for this modified set. Since all the leaves with modified weights are arranged at height 4, then the tree in figure 1.(b) is an optimal code tree with restricted maximal height 4 for the original set of weights W.

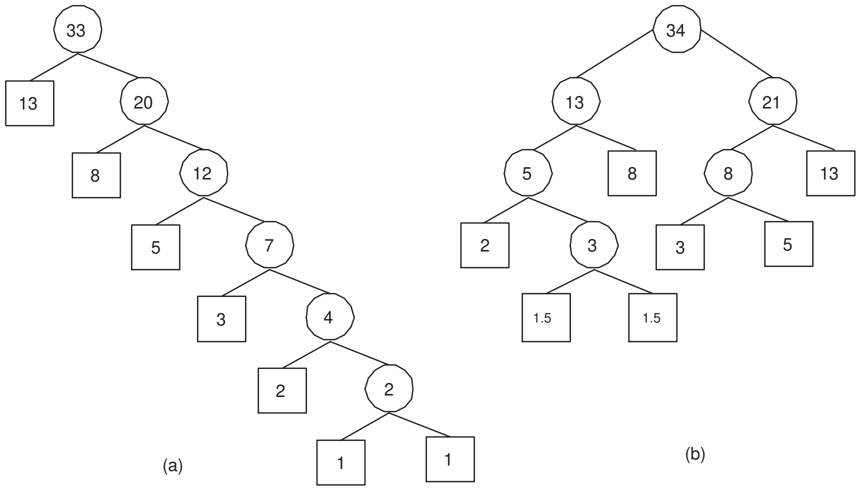

Figure 1: (a) A Huffman tree for the set of weights $\{1, 1, 2, 3, 5, 8, 13\}$. (b) An optimal tree with maximal restricted height 4 for the same set.

A natural question appears: *how to choose these multipliers ?*.

Our approach to choose these multipliers is to select a value x with $w_1 < x < w_n$ and set

$$\lambda_i = max\{0, x - w_i\}.$$

Then, we use a test to verify if the chosen set of multipliers, defined by x, is adequate. Let $\boldsymbol{w}^x$ be the new set of weights obtained through this perturbation. Observe that $\boldsymbol{w}^x = (max\{w_1, x\}, \ldots, max\{w_n, x\})$. Let T_x be a Huffman tree for the set $\boldsymbol{w}^x$. If $h(T_x) = L$, then we are close to the conditions of theorem 1. That is true because x is the lowest weight of set $\boldsymbol{w}^x$, and leaves with equal weights are always arranged in consecutive levels in a Huffman tree [Vit87], in this case levels L and $L - 1$.

One question arises: *how to select x ?*

The following theorem shows that if we increase the value of the parameter x, then the height of tree T_x cannot increase. This monotone property suggests that we can determine an adequate value for x through a convenient binary search.

Let us define T_x^- as a Huffman tree with minimum height for the set $\boldsymbol{w}^x$, and T_x^+ as a Huffman tree with maximum height for the same set. Schwartz [Sch64] shows how to construct both of these trees.

Theorem 2 *WARM-UP Theorem*

Let s, t be two real numbers with $0 < s < t$. Then, we have that $h(T_t^+) \leq h(T_s^-)$.

Let $\overline{\boldsymbol{h}}$ be the heights of the leaves of T_s^-, and let $\hat{\boldsymbol{h}}$ be the heights of the leaves of T_t^+. The optimality of $\overline{\boldsymbol{h}}$ and $\hat{\boldsymbol{h}}$ imply that

$$\sum_{i=1}^{n} max\{w_i, s\}.(\hat{h}_i - \overline{h}_i) \geq 0 \tag{3}$$

$$\sum_{i=1}^{n} max\{w_i, t\}.(\overline{h}_i - \hat{h}_i) \geq 0 \tag{4}$$

Let $\overline{k}$ be the greatest integer value such that $Max\{s, w_{\overline{k}}\} = s$ and let $\hat{k}$ be the greatest value such that $Max\{t, w_{\hat{k}}\} = t$. Without loss of generality, we can choose $\overline{\boldsymbol{h}}$ and $\hat{\boldsymbol{h}}$ such that $\overline{h}_i \geq \overline{h}_{i+1}$ and $\hat{h}_i \geq \hat{h}_{i+1}$ for $i = 1, 2, \cdots, n-1$.

Adding inequalities (3) and (4) we obtain that

$$(t-s).\sum_{i=1}^{\overline{k}}(\overline{h}_i - \hat{h}_i) + \sum_{i=\overline{k}+1}^{\hat{k}}(t - w_i)(\overline{h}_i - \hat{h}_i) \geq 0 \tag{5}$$

Since we have $\hat{k}$ equal weights in the tree T_t^+, it follows [Vit87] that $\hat{h}_1 \geq \hat{h}_i \geq \hat{h}_1 - 1$ for $i = 1, \cdots, \hat{k}$.

The highest height in an optimal tree is associated to the lowest weight. Hence, it follows that $\overline{h}_1 = h(T_s^-)$ and $\hat{h}_1 = h(T_t^+)$. Now, let us suppose for absurd that $\hat{h}_1 > \overline{h}_1$. In this case we have that $\hat{h}_i \geq \overline{h}_i$ for $i = 2, \cdots, \hat{k}$, because $\hat{h}_i \geq \hat{h}_1 - 1 \geq \overline{h}_1 \geq \overline{h}_i$. Under this condition, inequality (5) is not satisfied, what is an absurd. Therefore, we must have $h(T_t^+) \leq h(T_s^-)$.

2.3 Well Numbered Trees

The WARM-UP theorem allows us to search for an appropriate value for x by performing a binary search on the interval (w_1, w_n). If for a given x, $h(T_x^-) > L$, then we must increase x. On the other hand, if $h(T_x^+) < L$ we must decrease x. The special case where $h(T_x^-) < L$ and $h(T_x^+) > L$, is addressed in the next section.

Until now, we don't have any assurance that there exists x such that $h(T_x) = L$. Before we attempt to answer this question, let us define the concept of *well numbered trees*.

Definition 1 *A weighted full binary tree is well numbered if and only if*

(a) To each node is assigned a different integer number i, with $1 \leq i \leq 2n-1$;

(b) A parent node weight is equal to the sum of the weights of its two children nodes.

(c) If $i > j$, then the weight of the node with number i is not smaller than the weight of the node with number j;

(d) Let $i > j$. If the node with number i is son of a node with number i', and the node with number j is son of a node with number j', then $i' \geq j'$.

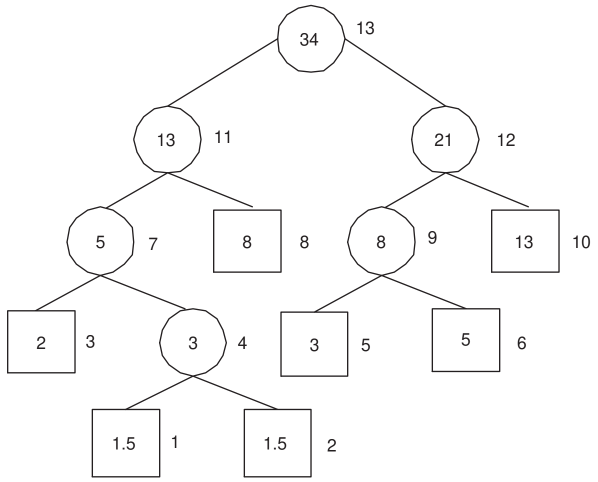

Figure 2: A well numbered tree

The tree in figure 2 is well numbered. Each number is drawn at the right side of the node. Now, we state three key properties concerning the well numbered trees.

Proposition 1 *Let $y_1, \ldots, y_{2n-1}$ be the nodes of a well numbered tree T. Furthermore, let y_{i_1} and y_{i_2}, with $i_2 > i_1$ be the two children of node y_i, and let y_{i_3} and y_{i_4} with $i_4 > i_3$ be the two children of node y_{i+1}. Under these assumptions, we have that $i_2 = i_1 + 1, i_3 = i_2 + 1$ and $i_4 = i_3 + 1$.*

First, we prove that $i_2 = i_1 + 1$. Assume that $i_2 > i_1 + 1$. Let z be the node with number $i_1 + 1$. It follows that the parent of z has number not smaller than the parent of y_{i_1} and number not greater than the parent of y_{i_2}. Hence, the parent of z has number i, what contradicts the fact that each internal node has only two children. The proof that $i_4 = i_3 + 1$ is analogous and the proof that $i_3 = i_2 + 1$ follows by the same arguments already used.

Proposition 2 *Let $y_1, \ldots, y_{2n-1}$ be the nodes of a well numbered tree T with corresponding heights $l_1, \ldots, l_{2n-1}$. Then, $0 \leq l_i - l_{i+1} \leq 1$ for $i = 1, \ldots, 2n-2$.*

First, let us prove that $l_{i+1} \leq l_i$ for $i = 1, \cdots, 2n-2$. It follows from properties (b) and (c) that y_{2n-1} is the heaviest node and hence the root of T. Hence, the result holds for $i = 2n-2$. Let us assume by induction that the result is valid for $i > p$ and then we prove it for $i = p$. Observe that the parent of y_p has number smaller or equal to the number of the parent of y_{p+1}. It follows by induction that the parent of y_p has a height not lower than the height of the parent of y_{p+1}. Since a parent's height is exactly one unity lower than the height of its children, the result holds for $i = p$.

Now, suppose that $l_i > l_{i+1} + 1$ for some i. In this case, the parent $y_{i'}$ of y_i has height $l_{i'} = l_i - 1$. Therefore, $i' \geq i+1$ and $l_{i'} = l_i - 1 > l_{i+1}$, what contradicts the inequality $l_{i+1} \leq l_i$ for $i = 1, \cdots, 2n-2$, that we have already proved. Hence, $l_i \leq l_{i+1} + 1$.

Proposition 3 *If T is well numbered, then T is a Huffman tree.*

It follows from proposition 1 that the tree T has the *sibling property*, defined by Gallager [Gal78]. Therefore, T is a Huffman tree.

2.4 The Stopping Condition

Now, we prove that there is a value x, such that $h(T_x) = L$. This proof gives an intuitive idea about the effects in the tree generated by changing the parameter x.

Theorem 3 *If $h(T^-_{w_1}) > L \geq \lceil \log n \rceil$ then there is a rational number $x = p/q$, with $q \leq n$ and $w_1 < x < w_n$, such that $h(T_x) = L$.*

Observe that $T^-_{w_1}$ is the Huffman tree for the original set of weights, since $max\{w_1, w_i\} = w_i$ for $i = 1, \ldots, n$. Let us assume that $T^-_{w_1}$ is well numbered. In the section 3 we show how to construct such a tree. We use y_i to denote the node with number i, ϕ_i to denote the weight of node y_i and n_i to denote the number of descendent leaves of node y_i with weight x. We define $n_i = 1$ when y_i is a leaf with weight equal to x. We also define δ_i by $\delta_i = (\phi_{i+1} - \phi_i)/(n_i - n_{i+1})$.

The well numbered tree in figure 2 is a T_x tree, with $x = 1.5$. For this tree, we have $n_1 = n_2 = 1$, $n_4 = n_7 = n_{11} = n_{13} = 2$, and the others nodes have $n_i = 0$. We also have that $\phi_1 = 0$, $\phi_2 = 0.5$, $\phi_4 = 0$ and so on. Now, we prove that one of the trees obtained by the following algorithm is a T_x tree with $h(T_x) = L$.

$x \leftarrow w_1$

$T \leftarrow T_x^-$

`Repeat`

  `Let` $I = \{i | i =$ `arg min` $\{\delta_i | n_i > n_{i+1}\}\}$

  `Define` i^* `to be the smallest integer in` I

  `For each leaf` y_j `of` T `with` $\phi_j = x$ `do`

    $\phi_j \leftarrow x + \delta_{i^*}$

  `Update all internal node weights to assure that a parent weight is equal to the sum of its children weights.`

  `In` T `interchange`[2] `the subtrees with roots` y_{i^*} `and` y_{i^*+1}

  `Assign number` i^* `to` y_{i^*+1} `and number` $i^* + 1$ `to` y_{i^*}

  $x \leftarrow x + \delta_{i^*}$

`Until` $I = \emptyset$

We divide the algorithm correctness proof into six parts.

(Part a) All trees constructed by the algorithm are well numbered.

items (a) and (b) of definition 1 are clearly respected by all trees T. Hence, we need to prove that items (c) and (d) are also respected.

For the initial tree, the result follows from the hypothesis under $T_{w_1}^-$. When the algorithm adds δ_{i^*} to the leaves with weight x and after updating the internal node weights, we still have $\phi_j \leq \phi_{j+1}$ for $j = 1 \ldots, 2n-2$, since δ_{i^*} is the minimum value necessary to produce a tie between the weights of two nodes with consecutive numbers. Therefore, items (c) and (d) of definition 1 hold. On the other hand, by interchanging the two chosen subtrees and also its corresponding root node numbers will also ensure items (c) and (d).

(Part b) In each cycle, the tree T is a T_x tree for the current value of x.

It follows from (Part a) and proposition 1 that each tree T is a Huffman tree. Thus, we must prove that the weights of the leaves of T are given by $max\{w_1, x\}, \ldots,$ $max\{w_n, x\}$ for the current value of x. The leaves of the initial tree have these weights. Let T^m and x^m be respectively, the tree T at the beginning of the mth cycle and the value of x at the beginning of this same cycle. We assume that the weights of the

[2] When two subtrees are interchanged, the second subtree becomes a child of the root parent of the first subtree and vice-versa.

leaves of T^m are given by $max\{w_1, x^m\}, \ldots,\ max\{w_n, x^m\}$ and then we prove that the weights of the leaves of T^{m+1} are given by $max\{w_1, x^{m+1}\}, \ldots, max\{w_n, x^{m+1}\}$.

We can suppose that $w_i \leq x^m < w_{i+1}$. If $x^{m+1} \leq w_{i+1}$, then the result clearly holds. On the other hand, if $x^{m+1} > w_{i+1}$ the result doesn't hold, since T^{m+1} would have a leaf with weight $w_{i+1} < x^{m+1}$. However, if $x^{m+1} > w_{i+1}$, then the integer i^* chosen during the mth cycle is such that $\delta_{i^*} = (x^{m+1} - x^m) > (w_{i+1} - x^m)$. Let y_j be the node of tree T^m such that $\phi_{j+1} = w_{i+1}$. Since $\phi_j \geq x_m$, we have that $\delta_j \leq w_{i+1} - x^m$. It follows that i^* would not belong to I and as a consequence the case $x^{m+1} > w_{i+1}$ is not possible.

(Part c) Node y_{i^*+1} is always a leaf.

First, let us assume that y_{i^*} is a leaf and that y_{i^*+1} is an internal node. In this case $n_{i^*} = 1$, otherwise i^* would not have been selected because $n_{i^*} = 0 \leq n_{i^*+1}$. As a consequence we get that $\phi_{i^*} = x$. The two children of y_{i^*+1} have numbers no greater than i^*. Therefore, these children must have weights x, otherwise we contradict either item (c) of definition 1 or the result of (Part b). Hence, $n_{i^*} = 1 < 2 = n_{i^*+1}$ and i^* would not belong to set I.

Now, let us assume that y_{i^*} and y_{i^*+1} are both internal nodes. Let $y_{i'}$, $y_{i'+1}$ be the sons of y_{i^*} and $y_{i'+2}$, $y_{i'+3}$ be the sons of y_{i^*+1}. It follows from the definition of δ_i that $\phi_{i'} + \phi_{i'+1} + (n_{i'} + n_{i'+1}).\delta_{i^*} = \phi_{i^*} + \delta_{i^*}.n_{i^*} = \phi_{i^*+1} + \delta_{i^*}.n_{i^*+1} = \phi_{i'+2} + \phi_{i'+3} + (n_{i'+2} + n_{i'+3}).\delta_{i^*}$. We also have that $\phi_{i'} + \delta_{i^*}.n_{i'} \leq \phi_{i'+1} + \delta_{i^*}.n_{i'+1} \leq \phi_{i'+2} + \delta_{i^*}.n_{i'+2} \leq \phi_{i'+3} + \delta_{i^*}.n_{i'+3}$, otherwise there would be a node y_j with $n_j > n_{j+1}$ such that $\delta_j < \delta_{i^*}$. To satisfy the two previous expressions we must have that $\phi_{i'} + \delta_{i^*}.n_{i'} = \phi_{i'+1} + \delta_{i^*}.n_{i'+1} = \phi_{i'+2} + \delta_{i^*}.n_{i'+2} = \phi_{i'+3} + \delta_{i^*}.n_{i'+3}$. Since $n_{i^*} > n_{i^*+1}$, $n_{i^*} = n_{i'} + n_{i'+1}$ and $n_{i^*+1} = n_{i'+2} + n_{i'+3}$, then either $n_{i'} > n_{i'+1}$ or $n_{i'+1} > n_{i'+2}$ or $n_{i'+2} > n_{i'+3}$. Hence, there is an index j with $i' \leq j < i' + 3 \leq i$, such that $n_j > n_{j+1}$ and $\delta_{i^*} = \delta_j$. This contradiction establishes the result.

(Part d) Let M be the sum of the numbers assigned to the leaves of T. The five statements below are true:

(i) $(n^2 - n)/2 \leq M \leq (3n^2 - n)/2$;

(ii) If the algorithm halts, then $M = (n^2 - n)/2$;

(iii) After each cycle the value of M either remains the same or decreases by exactly one unity;

(iv) In no more than $n^2 + n$ cycles the value of M satisfies $M = (n^2 - n)/2$;

(v) If $M = (n^2 - n)/2$, then $h(T) = \lceil \log n \rceil$.

Demonstrations:

(i) There are n leaves in T, each leaf assigned with a different number. Thus, $M \geq \sum_{j=1}^{n} j = (n^2 - n)/2$ and $M \leq \sum_{j=n}^{2n-1} j = (3n^2 - n)/2$.

(ii) In order to prove this item, we just need to show that $I \neq \emptyset$ whenever $M > (n^2 - n)/2$. If the sum of the numbers assigned to the leaves of T is greater than $n(n-1)/2$, then there is a leaf with number k, such that $k > n$. Since each number in the set $\{1, \ldots, 2n-1\}$ is a assigned to a node, then there is a node with number smaller than n.

It follows from item (d) of definition 1 that the parent of y_1 is the internal node with minimum number. Let m be the number of y_1's parent. We have that $m < n < k$ and $n_m \geq 1$. Since $\phi_k \geq \phi_m > x$ and y_k is a leaf, then $n_k = 0$. As $n_m > n_k$, there exist an index i with $m \leq i < k$, such that $n_i > n_{i+1}$. Hence, the set I is not empty.

(iii) It follow from Part (c) that y_{i^*+1} is a leaf. If y_{i^*} and y_{i^*+1} are leaves, then the value of M remains the same. On the other hand, if y_{i^*} is an internal node, then the value of M decreases by exactly one unity, since we interchange the numbers.

(iv) First, we show that in no more than n cycles y_{i^*} is a leaf. If y_{i^*} is a leaf then $\phi_{i^*} = x$, otherwise $n_{i^*} = 0 \leq n_{i^*+1}$. Furthermore, $\phi_{i^*+1} > x$, otherwise $n_{i^*+1} = 1 = n_{i^*}$. In this case the value of x is updated to $x + \phi_{i^*+1} - \phi_{i^*} = \phi_{i^*+1}$. Hence, the number of leaves with weight x is increased by one unity. Therefore, this case can happen in no more than n cycles, otherwise we would obtain a tree with more than n leaves.

When y_{i^*} is an internal node, the value of M decreases by one unity. Since $(n^2 - n)/2 \leq M \leq (3n^2 - n)/2$, then this second case can happen no more than $(3n^2 - n)/2 - (n^2 - n)/2 = n^2$. Therefore, in no more than $n^2 + n$ the value of M satisfies $M = (n^2 - n)/2$.

(v) If $M = (n^2 - n)/2$, then the numbers of all the leaves are not greater than n. Let l_j be the height of a generic leaf y_j. We have that the parent of y_1 has height $l_1 - 1$ and number greater than n. It follows from proposition 2 that the height of any leaf y_j satisfies $l_1 - 1 \leq l_j \leq l_1$. Therefore, all the leaves of T are arranged in consecutive heights. Hence, $h(T) = \lceil \log n \rceil$.

(Part e) On each cycle the height of T either remains the same or decreases by one unity.

If y_{i^*} has the same height as y_{i^*+1}, then the interchange does not modify $h(T)$. On the other hand, if the height of y_{i^*+1} is smaller than that of y_{i^*} by one unity, then the interchange either decreases $h(T)$ by one unity or does not modify $h(T)$, because y_{i^*+1} is a leaf.

(Part f) The variable x assumes only rational values of the form p/q with $p, q \in N$ and $1 \leq q \leq n$ during the algorithm execution.

Let x_m be the value of x at the beginning of the mth cycle. The first value of x is w_1, so x_1 is an integer. We observe that in the next assignment x assumes the value $x_m + ((\phi_{i^*+1} - \phi_{i^*})/n_{i^*}) = x_m + (\phi_{i^*+1} - A - x_m.n_{i^*})/n_{i^*} = (\phi_{i^*+1} - A)/n_{i^*}$, where A is the sum of the weights of the leaves that descend from y_{i^*} with corresponding weights values different from x_m. Since $\phi_{i^*+1} - A$ and n_{i^*} are positive integers and $1 \leq n_{i^*} \leq n$, we get that x assumes only values in the form p/q with $p, q \in$N and q $\leq$ n.

Algorithm Correctness

It follows from (Part d), that the algorithm can only stop if $M = (n^2 - n)/2$. On the other hand if $M = (n^2 - n)/2$, then $h(T) = \lceil \log n \rceil$. Since the first tree has height greater than L and after each cycle the height of T does not decrease by more than one unity, then the algorithm must obtain a tree T_x with height exactly L in no more than $n^2 + n$. Furthermore, (Part f) assures that x is a rational number p/q with $q \leq n$.

This result allows us to stop searching for a value x when the length of the search interval is smaller than $1/n^2$, since there is only one rational with denominator less than n in such interval, and it can be easily determined.

3 The Procedure Ties

Before describing our algorithm we must consider first a special case, the one where

$$h(T_x^-) < L \text{ and } h(T_x^+) > L.$$

For the same set of weights it is possible to exist more than one Huffman tree. This happens because when constructing the tree, Huffman's algorithm faces alternative choices on how to choose a node when there is more than one node with minimal weight.

Searching for an adequate value for x, we can face a situation where $h(T_x^-) < L$ and $h(T_x^+) > L$. If we increase x, then the new tree gets a height smaller than L. On the other hand, if we decrease the value of x, then we obtain a tree with height greater than L. Theorem 2 guarantees this fact. Since we have already shown that there is always a value x such that $h(T_x) = L$, then the current value of x is the one we are looking for. In this case, we can construct a Huffman tree with height exactly L for the current set of weights.

The trees in figure 3 are T_3 trees for the weights $1, 2, 4, 6, 10, 16, 26$. The left tree is a T_3^+ tree and its height is 6, while the right tree is a T_3^- tree and its height is 4. If we have been trying to find a T_x tree with height 5, we would face the special case adressed in this section.

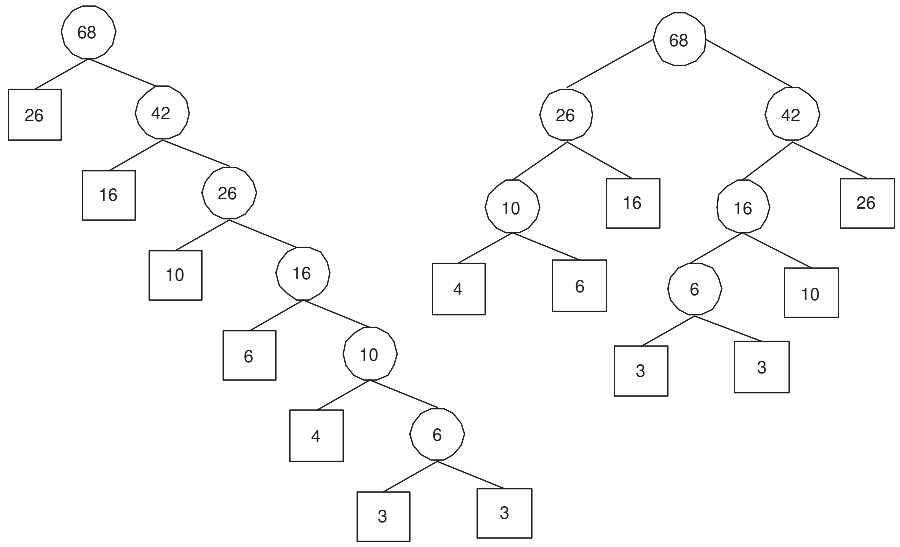

Figure 3: A T_3^+ tree and a T_3^- tree for the weights $1, 2, 4, 6, 10, 16, 26$.

Let us consider the well known [Lee76] implementation of Huffman's algorithm that uses a stack S to store the leaves of the tree, and a queue Q to store the internal nodes. At start, Q is empty and S contains the leaves of the tree sorted by weights, such that the leaf with lower weight is located at the top of S.

During the main step of this implementation, the weight of the leaf at the top of S is compared to the weight of the internal node at the head of Q. The node with smaller weight is selected. After the selection of two nodes, a new internal node is created. This new node is the parent of the selected nodes, and it is inserted at the tail of Q. Observe that whenever a node is selected, it is removed either from the top of S or from the head of Q. The main step is executed until S is empty and Q contains only one node.

We say that a tie occurs whenever a leaf at the top of S has the same weight as an internal node at the head of Q. If we choose a leaf whenever a tie occurs, we obtain a Huffman tree with minimum height. On the other hand, if we always choose an internal node, we get a Huffman tree with maximum height [Sch64].

3.1 Procedure Ties Description

Let T^- be a Huffman tree with minimum height and let T^+ be a Huffman tree with maximum height. Both of these trees having the same set of weights. The procedure Ties constructs a Huffman tree with height L when $h(T^-) < L$ and $h(T^+) > L$.

The procedure performs a binary search on the integers $1, \ldots, 2n-2$. For each selected integer k, it constructs a Huffman tree T^k using the implementation described above with an additional refinement: *choose a leaf on each one of the first k ties and then choose only internal nodes on all subsequent ties.* If it obtains the tree with height L, then the procedure ends. If the obtained tree has height lower than L, then the binary search proceeds with a decreased value of k. Otherwise, the binary search proceeds with an increased value of k.

3.2 Ties Correctness

Let us suppose that procedure Ties produces also a numbering of the nodes as follows. Number 1 is assigned to the first node removed either from the top of S or from the head of Q. Number 2 is assigned to the second removed node, and so on. It is easy to see that the constructed tree is well numbered.

Now, let y_i denote the node with number i in T^k, with corresponding weight and height given by ϕ_i and l_i respectively. Let us also assume that p is the number of the internal node selected when the $(k+1)$-th tie occurs, and that q is the number of the first leaf selected after node y_p. Observe that $\phi_p = \phi_{p+1} = \ldots = \phi_q$ since the $(k+1)$-th tie was between nodes y_p and y_q and T^k is well numbered. Furthermore, it follows from property 2 that $l_p \geq l_{p+1} \ldots \geq l_q \geq l_p - 1$. Under these assumptions, the following steps are performed to obtain T^{k+1} from T^k.

```
For i = q down to p + 1
  Interchange the subtree rooted at y_i with the one rooted at y_{i-1}
  Assign number i to node y_{i-1}, and number i - 1 to node y_i
Next
```

Two cases can occur when the procedure interchanges the subtrees. If the leaf y_i

has the same height as the internal node y_{i-1}, then the tree height is not affected. Otherwise, y_i has a height smaller than that of y_{i-1} by exactly one unity. In this second case, the interchange either reduces the tree height by one unity or keeps it the same. Since $l_p \geq \ldots \geq l_q \geq l_p - 1$, then the second case can only happen once. Therefore, we have that $h(T^{k+1}) \leq h(T^k) \leq h(T^{k+1}) + 1$.

When $h(T^{2n-1}) < L$ and $h(T^0) > L$, one can find a value k such that $h(T^k) = L$, by performing a binary search on $1, \ldots, 2n$. Hence, the procedure Ties can be used to construct a T_x tree with $h(T_x) = L$, since $h(T_x^-) < L$ and $h(T_x^+) > L$.

3.3 Ties Complexity

The binary search tries no more than $\lceil \log(2n-2) \rceil$ values for k. As we spend $O(n)$ to construct an optimal tree if the frequencies are already sorted, we have that the procedure spends an $O(n \log n)$ time in the worst case.

4 The WARM-UP ALGORITHM

Finally, we describe the algorithm. A pseudocode for it is given below.

```
WARM-UP Algorithm

Phase 0 Trying a Huffman Tree
      If h(T⁻_{w1}) ≤ L then Return(T⁻_{w1})

Phase 1 Binary search
      inf ← w1;   sup ← wn
      While sup-inf > 1/n² do
        x ← (inf+sup)/2
        If h(T⁻_x) = L then Return(T⁻_x)
        If h(T⁺_x) = L then Return(T⁺_x)
        If h(T⁻_x) < L and h(T⁺_x) > L then Return(Ties(x))
        If h(T⁻_x) > L then inf ← x
                    else sup ← x
      End While

Phase 2 Search for p/q
      q ← 1;   p ← ⌈q.inf⌉
```

```
While (p = q.inf or p ≥ q.sup) do
   q ← q + 1
   p ← ⌈q.inf⌉
End While
Return(Ties(p/q))
```

Observe that the algorithm first constructs a Huffman tree for the given set of weights. If this tree has height not greater than L, then the algorithm halts since this tree solves our problem. Otherwise, the algorithm performs a binary search on the interval (w_1, w_n). For each selected value x, it constructs the Huffman trees T_x^- and T_x^+. If either $h(T_x^-) = L$ or $h(T_x^+) = L$ then the algorithm halts. If that doesn't happen, then one of the following three actions is necessary:

(i) if $h(T_x^-) > L$ then increase the value of x and continue with the binary search;

(ii) if $h(T_x^+) < L$ then decrease the value of x and continue with the binary search;

(iii) if $h(T_x^+) > L$ and $h(T_x^-) < L$ then call the procedure Ties for the current set of weights.

The binary search also stops when the length of the searching interval is smaller than $1/n^2$. Whenever this happens, the algorithm finds the unique rational number p/q with $q \leq n$ in the last searching interval. Using this value of x, the algorithm builds a T_x tree. This follows immediately from theorem 3.

4.1 Approximation Error

In the previous section we proposed a method to construct a prefix code tree T_x with height L. This tree is not necessarily an optimal prefix code tree with restricted maximal height L. Now, we give an upper bound for the error of the code induced by this tree. Let us define this error ϵ by

$$\epsilon = (\sum_{i=1}^{n} p_i l_i - \sum_{i=1}^{n} p_i \overline{l_i}),$$

where $\overline{l}_i$ is the height of ith leaf of an optimal prefix code tree with restricted maximal height L, l_i is the height of ith leaf of the tree T_x constructed by WARM-UP ALGORITHM and p_i is the probability of symbol a_i, that is given by $w_i/(\sum_{j=1}^{n} w_j)$. The value $\sum_{i=1}^{n} p_i l_i$ is the *average code length*, and it represents the average number of bits used per symbol. In order to bound ϵ, we state an useful lemma.

Lemma 1 *Consider the weights $w_1 \leq \ldots \leq w_n$. For these weights, let $l_1, \ldots, l_n$ be the lengths of the code induced by a T_x tree with height L. Let also, $l_1^{L-1}, \ldots, l_n^{L-1}$ be lengths of the optimum prefix code with restricted maximal length $L-1$. Thus, we have that $\sum_{i=1}^{n} p_i l_i \leq \sum_{i=1}^{n} p_i l_i^{L-1}$.*

Let p be the largest integer, such that $l_p = L$. Consider problem P_L with the additional constrain that $l_i \leq L-1$ for $i = p+1, \ldots, n$. Similarly to theorem 1, one can show that the leaf heights of a T_x tree with height L solve the modified problem. Hence, we must have $\sum_{i=1}^{n} w_i l_i \leq \sum_{i=1}^{n} w_i l_i^{L-1}$. Otherwise, $l_1, \ldots, l_n$ would not be the optimal solution for this modified problem. Since $p_i = w_i / \sum_{i=1}^{n} w_i$, then we obtain that $\sum_{i=1}^{n} p_i l_i \leq \sum_{i=1}^{n} p_i l_i^{L-1}$

Now, we are ready to state the theorem that gives an upper bound for ϵ.

Theorem 4 *The maximum average code length difference ϵ between the code induced by the WARM-UP ALGORITHM and the optimal prefix code with restricted maximum length L is such that*

$$\epsilon < 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2} \textit{ for } L > \lceil \log n \rceil + 1,$$

where ψ is the golden ratio 1.618.

Milidiu and Laber [Mil97] show that if $L \geq \lceil \log n \rceil + 1$, then the maximum average code length difference between the the optimal prefix code with restricted maximal length L and the unrestricted optimal prefix code is smaller than $1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 1}$. Hence, it follows from this result and from lemma 1 that $\epsilon < 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2}$ for $L > \lceil \log n \rceil + 1$.

This result shows that the average code length difference is quite small for values of L not too close to $\lceil \log n \rceil$. For example, if $L - \lceil \log n \rceil \geq 15$, then the difference ϵ is less than 0.01. For $L - \lceil \log n \rceil \geq 20$, we have that $\epsilon < 0.0003$. Nevertheless, undesirable results like the difference being greater than 1 could occur. However, that would only happen if $L \leq \lceil \log n \rceil + 1$.

4.2 Complexity

In Phase 0, the algorithm spends an $O(n)$ time to construct the Huffman tree since the weights are already sorted.

In Phase 1, the algorithm can spend an $O(n \log(n^2.w_n))$ time in the worst case. That is true because in every cycle the length of the search interval is divided by

2. Hence, in no more than $\lceil \log n^2.(w_n - w_1) \rceil$ steps we have $\mathtt{sup} - \mathtt{inf} \leq \mathtt{1/n^2}$. Moreover, during each cycle we spend an $O(n)$ time to determine both a Huffman tree with minimum height and a Huffman tree with maximum height, for the current set of weights. In phase 1, the algorithm can also execute one call to procedure Ties. However, that does not affect the time complexity, since it represents an additional effort bounded by $O(n \log n)$.

If the algorithm reaches Phase 2, it spends an $O(n)$ time to find the adequate rational number p/q. Therefore, we conclude that the WARM-UP ALGORITHM has an $O(n \log n + n \log w_n)$ worst case time complexity.

By using Moffat's approach [Mof97], the WARM-UP time complexity can be easily reduced to $O(n \log n + r \log(n/r) \log \overline{w})$, where r is the number of distinct weights in the given set of weights.

4.3 Limitations

Theorem 1 shows that if all the leaves with weight x in a T_x tree have height L, then T_x is an optimal code tree with restricted maximal height L. Unfortunately, for some sets of weights there is no such x. For instance, consider $L = 3$ and the set of weights given by $\{1, 1, 1, 13, 15\}$. Figure 4.(a) shows the tree topology for all T_x with $1 \leq x \leq 5$, and figure 4.(b) shows the tree topology for all T_x with $5 \leq x \leq 15$. Observe that, in the tree topology of picture (b), not all the leaves with weight less than 5 have height 3. Moreover, the average code length for this tree is 2.064. On the other hand, this measure is equal to 2.032 for the optimal code tree exhibited in figure 4.(c).

One could ask what happens, if we choose the multipliers in a different way. For the set of weights presented in figure 4, it has been proved [Lab97] that the optimal code tree can not be obtained, even though we try all the sets of multipliers.

5 Conclusions

The algorithm introduced in this paper is very easy to implement, since it consists of successive constructions of Huffman trees. Furthermore, the experiments reported in [Mil97b] suggest that the WARM-UP algorithm is a very good choice when constructing length restricted prefix codes for large alphabets, if we consider time and space requirements.

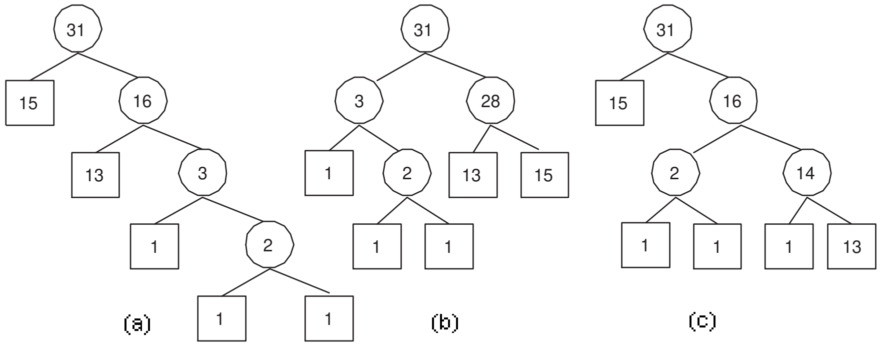

Figure 4: (a) Tree T_x for the set of weights $W = \{1, 1, 2, 3, 5, 8, 13\}$ and $1 \leq x \leq 5$. (b) Tree T_x for the set of weights W and $5 \leq x \leq 15$. (c) An optimal tree with restricted maximal height 3 for the set W.

The value $\log w_n$ that appears in the worst case complexity is lower than 40 ($w_n \approx 10^{12}$) for practical purposes and the constants involved in the complexity are smalls, since there is a lot of very optimized implementations of Huffman trees. In order to improve its time efficiency, the algorithm can be executed until the average code length difference ϵ is smaller than a given acceptable error δ. An upper bound for ϵ can be obtained after the construction of each tree T_x by using the value of the dual function.

Furthermore, we believe that the techniques developed here can be useful to address the problem of constructing optimal prefix codes satisfying other kinds of constrains.

References

[Agg94] Aggarwal, A., Schieber B., Tokuyama T., *Finding a minimum weight K-link path in graphs with Monge property and Applications*, Journal of Discrete and Computational Geometry 12(1994).

[Ahu93] Ahuja, R. K., Magnanti, T. L., Orlin J. B., *Network Flows:theory, algorithm and applications*, Prentice Hall, 1993.

[Cho85] Choueka, Y., Klein S. T. and Perl, Y., *Efficient variants of Huffman Codes in high level languages*, Proc 8th ACM-SIGIR Conf, Montreal, 122-130 (1985).

[Eve79] Even, S., *Graph Algorithms.* Computer Science Press, 1979.

[Fra93] Fraenkel, A.S., and Klein, S. T., *Bounding the depth of search trees*, The Computer Journal, 36, (7), 668-678 (1993).

[Gal78] Gallager, R. G., *Variations on a theme by Huffman*, IEEE Transactions on Information Theory., IT-24, 668-674, (1978).

[Gar74] Garey M. R., *Optimal binary search trees with restricted maximal depth*, SIAM J. Computing 23(1974), 101-110.

[Gil71] Gilbert, E.N., *Codes based on innacurate source probabilities*, IEEE Transactions on Information Theory., IT-17, 304-314, (1971).

[HuT72] Hu, T. C., and Tan, K. C., *Path length of binary search trees.*, SIAM J. Appl Math 22 (1972), 225-234.

[Huf52] Huffman, D.A., *A Method for the Construction of the Minimum Redundancy Codes*, Proc. Inst. Radio Engineers, Vol 40,No 10, 1098-1101, September 1952.

[Kat76] Katona, G. O. H., and Nemetz O. H. T., *Huffman codes and self information*, IEEE Transactions on Information Theory., IT-22, 337-340, (1976).

[Ktj95] Katajainen, J., Moffat, A., *Inplace calculation of minimum-redundancy codes*, In Proc. Workshop an Algorithms and Data Structures, Kingston University, Canada, August 1995. Springer-Verlag.

[Lab97] Laber, E. S., Um Algoritmo Eficiente para Construção de Códigos de Huffman com Restrição de Comprimento, M.Sc. Dissertation, Departamento de Informática, PUC-RJ, Brasil, 1997.

[Lar90] Larmore, L.L. and Hirschberg, D.S., *A Fast Algorithm for Optimal Lenght-Limited Huffman Codes*, Journal of the ACM, July (1990), 464-473.

[Lar95] Larmore, L.L. and Przytycka, T.M., *Constructing Huffman Trees in Parallel*, Siam Journal on Computing, Novemeber (1995), 1163-1169.

[Lee76] Leeuwen, J. van (1976), *On the construction of Huffman trees*, In proc. 3rd Int. Coll. On Automata, Languages, and Programming, pp 382-410.

[Meg83] Megiddo, N., *Applying Parallel Computation Algorithms in the Design of Serial Algorithms* Journal of ACM 30 (1983).

[Mil97] Milidiu, R.L. and Laber, E.S., *Improved Bounds on the Inefficiency of Length-Restricted Prefix Codes*, Monografias em Ciência da Computação 33/97, Departamento de Informática, PUC-RJ, Brasil, 1997.

[Mil97b] Milidiú, R.L., Pessoa A. A. and Laber, E.S., *Practical use of the WARM-UP Algorithm on Length-Restricted Codes*, To appear in the Proceedings of the Fourth Latin American Workshop on String Processing, Chile, 1997.

[Min93] Minoux, M., *Programmation mathematique, theorie et algorithmes tome 1*, Collection Technique et Scientifique des Telecomunications, Dunod (1993).

[Mof97] Moffat A. and Turpin A. *Efficient Construction of Minimum-Redundancy Codes For Large Alphabets*, To appear in IEEE Transactions on Information Theory.

[Sch64] Schwartz, E. S., *An Optimum Encoding with Minimum Longest Code and Total Number of Digits*, Information and Control 7, 37-44 (1964).

[Scb95] Schieber B. *Computing a minimum-weight k-link path in graphs with the concave monge property.*, In Proc. 6th ACM IEEE Symposium on Discrete Algorithms, pages 405-411,San Francisco, January 1995.

[Tur95] Turpin A. and Moffat A., *Practical Length-limited Coding for Large Alphabets*, The Computer Journal, 38, (5), 339-347 (1995).

[Vit87] Vitter J. S., *Design and Analysis of Dynamic Huffman codes*, Journal of ACM 34, 4, Oct, 825-845.

[Voo74] Van Voorish D. C., *Constructing codes with bounded codeword lengths*, IEEE Transactions on Information Theory., IT-20, 288-290, (1974).

[Zob95] Zobel, J. and Moffat, A. *Adding Compression to full-text retriweval system.* Software Practice and Experience, 25(8): 891-903, 1995.