A Note on the Construction of Error Detecting/Correcting Prefix Codes

Artur Alves Pessoa
artur@producao.uff.br
Production Engineering Department
Universidade Federal Fluminense, Brazil
March 7, 2007

Keywords: prefix codes, approximation algorithms, data compression, information retrieval

1 Introduction

Encoding schemes designed for efficient data communication through unreliable channels shall provide at least two basic features: compression and error detection/correction. While the provided compression shall be as efficient as possible (for the given computational resource constraints), error correction requirements depend on the reliability of the communication channel. One possible error correction scheme consists of adding error detection capabilities to the code and then retransmitting every data package where an error is found. In this case, the earlier detection of an error reduces the amount of data that must be retransmitted. Typically, compression and error detection/correction capabilities are independently added to the encoded messages [3]. On the other hand, designing a code with maximum compression efficiency subject to some error detection/correction requirement leads to interesting combinatorial optimization problems. The same capabilities are also required for encoding schemes used to store/retrieve data to/from unreliable media. In this case, an additional complication is that intermediate portions of the encoded message may be randomly retrieved. For that, there exists suitable compression schemes that support the decode of a piece of data without the need to decompress the message from the beginning [6, 9]. Hence, a natural question that arises
is whether error detection/correction capabilities can be embedded in such compression schemes.

In this context, we define a \( k \)-bit Hamming prefix code as a binary code with the following property: for any codeword \( x \) and any prefix \( y \) of another codeword, both \( x \) and \( y \) having the same length, the Hamming distance between \( x \) and \( y \) is at least \( k \). Given an alphabet \( A = [a_1, \ldots, a_n] \) with corresponding probabilities \( [p_1, \ldots, p_n] \), the \( k \)-bit Hamming prefix code problem is to find a \( k \)-bit Hamming prefix code for \( A \) with minimum average codeword length \( \sum_{i=1}^{n} p_i \ell_i \), where \( \ell_i \) is the length of the codeword assigned to \( a_i \). Note that the well-known prefix code problem is a special case of the previous problem, where \( k = 1 \). In this case, the optimal solution can be obtained by the Huffman’s algorithm [4] in \( O(n \log n) \) time. A prefix code constructed by the Huffman’s algorithm is usually referred to as a Huffman code. On the other hand, the Hamming distance is a useful tool to design codes with error detection/correction capabilities [3]. Any fixed-length code whose Hamming distance is \( k \) allows one to detect errors of at most \( k - 1 \) bits or correct errors of at most \( [(k - 1)/2] \) bits in each codeword. Moreover, some fixed-length codes with Hamming distance \( k \) are also easy to construct. Let \( d_k(n) \) be the number of additional bits that shall be added to a fixed-length code of \( n \) codewords in order to achieve a Hamming distance of \( k \). Note that \( d_2(n) = 1 \) since it suffices to add a parity bit at the end of each codeword to achieve a Hamming distance of 2. Moreover, it can be shown that \( d_3(n) = \Theta(\log \log n) \). In fact, Hamming devised a code whose Hamming distance is 3 that uses \( n = 2^{2^m-1} \) codewords of \( 2^m - 1 \) bits each, for \( m > 1 \). Such a code can be constructed in \( O(n \log n) \) time.

The \( k \)-bit Hamming prefix code problem is based on an example given by Hamming [3] that combines the compression provided by Huffman codes and the error protection provided by Hamming codes. Note that any \( k \)-bit Hamming prefix code allows one to detect errors of at most \( k - 1 \) bits or correct errors of at most \( [(k - 1)/2] \) bits during the decode of the first wrong codeword in spite of the different codeword lengths. We found only one paper in the literature addressing this specific problem [8], where the authors propose a heuristic for the 3-bit Hamming prefix code problem (referred to as ECCC problem). In this case, no worst-case bound on the compression loss is provided. In [7], Pinto et. al. give a polynomial algorithm for finding optimal prefix codes where all codewords have even parities (say \textit{Even prefix codes}). Although the authors have the same motivation, the resulting code only ensure the detection of (an even number of) errors at the end of the encoded message.

The following example illustrates the difference between the Even prefix codes and 2-bit Hamming prefix codes. Table 1 gives both an Even prefix code and a 2-bit Hamming prefix code for the letters of the word “BANANA”. When using the Even prefix code, “BANANA” is encoded as 1010110110. In this case,
if the first bit is changed to zero, then the resulting message (excluding the last two bits) is erroneously interpreted as “AABB”. In this case, the error is detected only when the last bit is decoded. On the other hand, “BANANA” is encoded as 1111001100001100 using the 2-bit Hamming prefix code of Table 1. In this case, if the first bit is changed to zero then the minimum Hamming distance between codewords and prefixes ensure that the prefix 01 of the resulting message is immediately interpreted as an error.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Even prefix code</th>
<th>2-bit Hamming prefix code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>N</td>
<td>11</td>
<td>1100</td>
</tr>
<tr>
<td>B</td>
<td>101</td>
<td>1111</td>
</tr>
</tbody>
</table>

Table 1: An Even prefix code and a 2-bit Hamming prefix code for the letters of the word “BANANA”.

As far as we know, no polynomial (exact or approximate) algorithm and no hardness proof has been found for the $k$-bit Hamming prefix code problem, for any value of $k$.

In this paper, we propose a general approximation algorithm for the $k$-bit Hamming prefix code problem. Let $\alpha_k$ be an $O(r_k(n))$-time algorithm for calculating fixed-length codes with Hamming distances $k$ whose codewords are $d_k(n)$ bits longer than $\lceil \log_2 n \rceil$. Our algorithm uses $\alpha_k$ to calculate an approximate $k$-bit Hamming prefix code in $O(r_k(n) + n \log n)$ time. In this case, we show that the average codeword length of the resulting code is at most $O(d_k(n) + \log^* n)$ bits longer than the average codeword length of a Huffman code. In our analysis, we assume that $d_k(n)$ is either constant or satisfies $d_k(\lceil \log_2 n \rceil + 1) \leq \delta d_k(n)$ for some positive $\delta < 1$ and $n$ sufficiently large.

This paper is organized as follows. In section 2, we introduce some useful properties of both prefix codes and $k$-bit Hamming prefix codes. In section 3, we describe and analyze our approximation algorithm. In section 4, we summarize our conclusions. Throughout this paper, let us use $\log x$ to denote $\log_2 x$.

## 2 Combinatorial Properties

It is well known that there is a one-to-one correspondence between prefix codes and binary trees [2], where each codeword in the former corresponds to a leaf in the latter. Such binary trees must be ordered in the sense that the left and the right children of a node are distinguished from each other. Throughout this paper, let us assume that all binary trees are ordered. The weighted external path length of a binary tree is the
weighted sum of its leaf levels, where each leaf is weighted with the probability of the corresponding symbol. Let us use \( c(T) \) and \( h(T) \) to denote the weighted external path length of \( T \) and the number of distinct leaf levels in \( T \), respectively. Since every \( k \)-bit Hamming prefix code is a special case of a prefix code, the binary tree representation can (and will) be used as well. Let us refer to binary trees that correspond to \( k \)-bit Hamming prefix codes as \( k \)-bit Hamming trees.

Figure 1: The 2-bit Hamming tree \( \bar{T}^2(4) \).

Throughout this paper, we use \( \bar{T}^k(n) \) to denote the \( k \)-bit Hamming tree that corresponds to a fixed-length code with \( n \) codewords. Figure 1 gives an example of \( \bar{T}^2(4) \). In this figure, circles and squares represent internal nodes and leaves, respectively. Note that all leaves are at level 3 in this tree, which corresponds to each codeword having exactly one extra bit in addition to the required \( \lceil \log n \rceil \) bits, that is, \( d_2(4) = 1 \). A strictly binary tree is a binary tree where every internal node has exactly two children. Every Huffman code (as well as most approximate ones) correspond to strictly binary trees. In contrast, no \( k \)-bit Hamming tree (for \( k > 1 \) and \( n > 1 \)) is strictly binary.

We consider a leaf \( x \) to be leftward (rightward) from another leaf \( y \) in a binary tree when \( x \) is visited before (after) \( y \) in an in-order traversal. A canonical tree is a binary tree where the leaf levels are non-increasing from left to right. Every binary tree can be rearranged so that it becomes a canonical tree without changing the leaf levels [1]. We say that a canonical tree \( T \) is shrinkable when, for every leaf level \( \ell \), there is a subtree \( T_\ell \) of \( T \) that contains all leaves at level \( \ell \) in \( T \) and no other leaf. In this case, shrinking \( T \) means replacing each subtree \( T_\ell \) by a single leaf with probability equal to the sum of the probabilities of the leaves in \( T_\ell \). The following proposition gives a useful property of canonical trees.

**Proposition 1** For every canonical tree \( T \), there exists a shrinkable canonical tree \( T' \) with \( c(T') \leq c(T) + 1 \) and \( h(T') \leq h(T) \).

**Proof:** Let \( \ell_1, \ldots, \ell_{h(T)} \) be the leaf levels of \( T \) in a non-decreasing order. Let also \( m_\ell \) be the number of leaves at level \( \ell_i \), for \( i = 1, \ldots, h(T) \). A crucial observation is that \( T \) can always be rearranged into a shrinkable
canonical tree without changing the leaf levels if $m_i$ is a power of two, for $i = 1, \ldots, h(T)$. Hence, then we give a procedure that iterates over the leaf levels of $T$ increasing the number of leaves at each level to the next power of two. Our procedure starts with an empty tree $T'$. Then, at the $i$th iteration, it places exactly $m'_i = 2^\left\lceil \log m_i \right\rceil$ leaves at level $\ell'_i = \ell_i + \left\lceil (m'_i - m_i)/m'_i \right\rceil$ in $T'$. When the number of remaining leaves $r$ is smaller than $m'_i$, the procedure adds $m'_i - r$ dummy leaves to complete the subtree of level $\ell'_i$ in $T'$ and stop. After that, all dummy leaves are removed. Since we always have $m'_i \geq m_i$ and $\ell'_i \leq \ell_i + 1$, we obtain that the level of each leaf in $T'$ is not greater than its level in $T$ plus one. Hence, we have $c(T') \leq c(T) + 1$.

Figure 2: (a) A canonical tree $T$; (b) a shrinkable canonical tree obtained from $T$

Figure 2 illustrates the procedure described in the proof of Proposition 1. In Figure 2.(a), an example of a canonical tree $T$ is given. The shrinkable canonical tree $T'$ obtained from $T$ is shown in Figure 2.(b). In this figure, the only inserted dummy leaf is drawn with dashed lines. Note that the subtree rooted at $x$ ($y$) contains all leaves placed at level 3 (4) in $T'$.

3 The Approximation Algorithm

In this section, we describe and analyze our approximation algorithm for the $k$-bit Hamming prefix code problem.

A pseudo-code for this algorithm is given in Table 2. In this pseudo-code, the function Construct receives a list of input leaf probabilities as an argument and returns an approximate $k$-bit Hamming tree. In Step 0, if the number $n$ of input probabilities is not greater than a given constant $K$, then an optimal $k$-bit Hamming tree is obtained by testing all possible trees. If otherwise $n > K$, then Step 1 is executed. In this step, the algorithm proposed in [5] is used to construct an approximate length-restricted prefix code $T_1$ such that $c(T_1)$ is at most 1 bit longer than the average codeword length of a Huffman code. In Step 2, a canonical shrinkable tree $T_2$ is obtained from $T_1$ using the procedure described in the proof of Proposition 1. In Steps 3 and 4, $T_2$ is shrunk and the leaf probabilities of the resulting tree $T_3$ are used in a recursive call to the
function \texttt{Construct}(p_1, \ldots, p_n) \\
\textbf{Step 0:} if \( n \leq K \), then return an optimal \( k \)-bit Hamming tree with \( n \) leaves; \\
\textbf{Step 1:} \( T_1 \leftarrow \) a binary tree with \( n \) leaves and height at most \( \lceil \log n \rceil + 1 \); \\
\textbf{Step 2:} \( T_2 \leftarrow \) a canonical shrinkable tree obtained from \( T_1 \); \\
\hspace{1cm} let \( m_i \) be the number of leaves at the \( i \)th leaf level in \( T_2 \), for \( i = 1, \ldots, h(T_2) \); \\
\textbf{Step 3:} \( T_3 \leftarrow \) the result of shrinking \( T_2 \); \\
\hspace{1cm} let \( p'_1, \ldots, p'_{h(T_2)} \) be the leaf probabilities of \( T_3 \); \\
\textbf{Step 4:} \( T_4 \leftarrow \texttt{Construct}(p'_1, \ldots, p'_{h(T_2)}) \); \\
\textbf{Step 5:} replace the \( i \)th leaf in \( T_4 \) by \( \bar{T}_k(m_i) \), for \( i = 1, \ldots, h(T_2) \); \\
\hspace{1cm} return the resulting tree; \\
\textbf{End Function} \\

Table 2: A pseudo-code for constructing approximate \( k \)-bit Hamming trees \\

function \texttt{Construct}. Finally, in Step 5, the leaves of the reduced \( k \)-bit Hamming tree obtained through the recursive call are replaced by subtrees that correspond to fixed-length codes whose Hamming distances are \( k \). Note that this operation preserves the Hamming distances between codewords and prefixes. 

3.1 Analysis 

Here, we prove an upper bound on the compression loss introduced by our algorithm. Later, we comment on its time complexity. 

Now, let us define the function \( \log^{(j)}_{x+1} x \) by the following recursion: 

\[
\log^{(j)}_{x+1} x = \begin{cases} 
  x, & \text{for } j = 0; \\
  \left\lfloor \log \left( \log^{(j-1)}_{x+1} x \right) \right\rfloor + 1, & \text{for } j > 0;
\end{cases}
\]

Then, we define the function \( \log^*_{x+1} x \) as the smallest integer \( j \) such that \( \log^{(j)}_{x+1} x \leq 3 \). Let \( \hat{T} \) be the \( k \)-bit Hamming tree returned by the function \texttt{Construct} and \( T^* \) be a Huffman tree for the same input probabilities. We have the following theorem. 

\textbf{Theorem 1} If \( K \geq 3 \), then \( c(\hat{T}) \leq c(T^*) + \log^*_{x+1} n \sum_{j=0}^{\log^*_{x+1} n} \left( d_k \left( \log^{(j)}_{x+1} n \right) + 2 \right) + O(1) \). 

\textbf{Proof:} We prove it by induction on the value of \( n \). If \( n \leq K \), then \( c(\hat{T}) - c(T^*) = O(1) \) since \( K = O(1) \).
In this section, we present some conclusions about the results of this paper.

4 Conclusions

Consider the intermediary trees constructed during the first call to the function Construct. Let also \( T^*_n \) be a Huffman tree for the probabilities \( p'_1, \ldots, p'_{h(T_2)} \) of the shrunk leaves of \( T_3 \). By inductive hypothesis, we have

\[
c(T_4) \leq c(T^*_n) + \sum_{j=0}^{\log_{k+1} h(T_2)} \left( d_k \left( \log_{j+1} h(T_2) \right) + 2 \right) + O(1).
\]

Moreover, we have

\[
c(T^*_n) + \sum_{i=1}^{h(T_2)} c(\bar{T}^k(m_i)) = c(T^*_n) + \sum_{i=1}^{h(T_2)} p'_i \left( \lceil \log m_i \rceil + d_k(m_i) \right) \leq c(T^*_n) + \sum_{i=1}^{h(T_2)} p'_i \left( \lceil \log m_i \rceil + d_k(n') \right) \leq c(T'_3) + \sum_{i=1}^{h(T_2)} p'_i \left( \lceil \log m_i \rceil + d_k(n') \right) = c(T_2) + d_k(n') \leq c(T_1) + 1 + d_k(n') \leq c(T^*_n) + 2 + d_k(n').
\]

Note that (1) holds because \( c(T^*_n) \leq c(T_3) \) since \( T^*_n \) is an optimal tree for the leaf probabilities \( p'_1, \ldots, p'_{h(T_2)} \).

Moreover, (2) and (3) follows from Proposition 1 and the approximation bound given in [5], respectively. Since \( c(\bar{T}) = c(T_4) + \sum_{i=1}^{h(T_2)} c(\bar{T}^k(m_i)) \), we obtain that

\[
c(\bar{T}) \leq c(T^*_n) + 2 + d_k(n') + \sum_{j=0}^{\log_{k+1} h(T_2)} \left( d_k \left( \log_{j+1} h(T_2) \right) + 2 \right) + O(1).
\]

Since \( h(T_2) \leq \log_{k+1} n' \), this theorem follows. \( \blacksquare \)

By the previous theorem, it is clear that \( c(\bar{T}) - c(T^*_n) = O(\log^* n) \) when \( d_k(n) = O(1) \), and that \( c(\bar{T}) - c(T^*_n) = O(d_k(n)) \) when \( d_k(\lceil \log_2 n \rceil + 1) \leq \delta d_k(n) \) for some positive \( \delta < 1 \) and \( n \) sufficiently large.

For the time complexity of our algorithm, we observe that the time spent by the first call to function Construct dominates that of the remaining recursive calls. This leads to an execution time of \( O(n \log n + r_k(n)) \), where the \( O(r_k(n)) \) term is due to the construction of \( \bar{T}^k(m_1), \ldots, \bar{T}^k(m_{h(T_2)}) \), in Step 5.

4 Conclusions

In this section, we present some conclusions about the results of this paper.
<table>
<thead>
<tr>
<th>Code type</th>
<th>Hamming distance</th>
<th>Added bits</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-length</td>
<td>2</td>
<td>1</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Prefix-free</td>
<td>2</td>
<td>$O(\log^* n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Fixed-length</td>
<td>3</td>
<td>$\Theta(\log \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Prefix-free</td>
<td>3</td>
<td>$O(\log \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

Table 3: Additional bits and running times required to achieve each Hamming distance.

Table 3 shows the additional bits required to achieve each Hamming distance for each type of code. In this table, the rows that correspond to prefix-free codes summarize the main results of this paper. The column Added bits contains upper bounds on the difference between the average codeword length of each code and the average codeword length of a Huffman code. For fixed-length codes, we assume uniform probabilities. For $k = 2$, the code constructed by our algorithm allows the detection of any one-bit error during the decode of each codeword. If we set $k = 3$, then the resulting code allows the correction of any one-bit error or the detection of any two-bit error in each codeword. Note that, for $k \geq 3$, the $O(\log^* n)$ additional bits introduced by our algorithm are asymptotically dominated by the additional bits required to achieve the desired Hamming distance with fixed-length codes.

An interesting variant of our algorithm is obtained when we use fixed-length codes with Hamming distance one in the first call to the function Construct and Hamming distance two in the further recursive calls. This leads to a code that allows the detection of any one-bit error during the decode of each codeword and the correction of the corresponding codeword length. As a result, the decode can continue (after skipping the wrong codeword) without loss of synchronism. It can be shown that such a code is at most $O(\log \log \log n)$ bits longer than the Huffman code.

References


