Exact and experimental algorithms for a Huffman-based error detecting code *

Paulo E. D. Pinto †
Fábio Protti †
Jayme L. Szwarcfiter §

February 23, 2006

Abstract

Even codes are binary prefix codes, in which each encoding contains an even number of 1’s. They are Huffman based codes with the additional property of being able to detect the occurrence of an odd number of 1-bit errors in the message. They have been defined motivated by a problem posed by Hamming in 1980. Even codes have been studied for the case in which the symbols have uniform probabilities. In the present work, it is considered the general situation of arbitrary probabilities. An exact algorithm for constructing an optimal even code is described. The algorithm has complexity $O(n^3)$, where $n$ is the number of symbols. Further, two heuristics for constructing nearly optimal even codes are presented, which require $O(n \log n)$ time. The cost of the even code constructed by the second heuristics is at most 16.7% higher than the cost of a Huffman code, for the same probabilities. However, computer experiments suggest that, for practical purposes, this value seems to be much less: about 5% or less, for $n$ large enough. This corresponds to the overhead in the size of the encoded message, for having the ability to detect errors. Concerning undetected errors, we obtain bounds on the probability of the trees to produce $k$ consecutive wrong symbols, which suggest that such probability is small, for large messages.

Keywords: Data compression, error detection, even codes, Huffman codes, Hamming codes.

*This work is partially supported by CNPq and FAPERJ
†DICC/IME/UERJ, Rio de Janeiro, RJ, Brazil. Email: paulo@ime.uerj.br
‡IM and NCE/UFRJ, CCMN, Caixa Postal 224, CEP 20001-970, Rio de Janeiro, RJ, Brazil. Email: fabio@nce.ufrj.br
§NCE and COPPE/UFRJ, Rio de Janeiro, RJ, Brazil. Email: jayme@nce.ufrj.br
1 Introduction

Huffman codes [1] form one of the most traditional methods of coding. One of the important aspects of these codes is the possibility of handling encodings of variable sizes. A great number of extensions and variations of the classical Huffman codes have been described through the time. For instance, Faller [2], Gallager [3], Knuth [4] and Milidiú, Laber and Pessoa [5] addressed adaption methods for the construction of Huffman trees. Huffman trees with minimum height were described by Schwartz [6]. The construction of Huffman type trees with length constraints was considered by Turpin and Moffat [7], Larmore and Hirschberg [8] and Milidiú and Laber [9, 10]. On the other hand, Hamming formulated algorithms for the construction of error detecting codes [11]. Further, Hamming [11] posed the problem of describing an algorithm that would combine advantages of Huffman codes with the noise protection of Hamming codes. The idea is to define a prefix code in which the encodings contain redundancies that allow the detection of certain kinds of errors. This is equivalent to forbid some encodings which, when present in the reception, would signal an error. Such a code is a Hamming-Huffman code, and its representing binary tree is a Hamming-Huffman tree. In a Huffman tree, all leaves correspond to encodings. In a Hamming-Huffman tree, there are encoding leaves and error leaves. Hitting an error leaf in the decoding process indicates the existence of an error. The problem posed by Hamming is to devise a method for detecting the occurrence of an error of one bit, as illustrated in the following example given by Hamming [11], p.76. Table 1 shows the symbols and their corresponding encodings. Figure 1 depicts the corresponding Hamming-Huffman tree. Error leaves are represented by black nodes. An error of one bit in the above encodings would lead to an error leaf, in the decoding process.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>000</td>
</tr>
<tr>
<td>b</td>
<td>0110</td>
</tr>
<tr>
<td>c</td>
<td>1010</td>
</tr>
<tr>
<td>d</td>
<td>1100</td>
</tr>
<tr>
<td>e</td>
<td>1111</td>
</tr>
</tbody>
</table>

Motivated by the above problem, we have recently proposed [12] a special prefix code, called even code, in which each encoding contains an even number of 1's. This code has the property of detecting the occurrence of any odd number of 1-bit errors in the message and the great majority of an even number of 1-bit errors, as we shall see in Section 5. It is many times more powerful in error detection than just appending one parity bit at the end of one encoded message.

In [12], the study was restricted to codes corresponding to symbols having uni-
form probabilities. The present work considers the general situation of arbitrary probabilities. An extended abstract containing part of these results appears in [13]. First, we describe an exact algorithm for constructing an optimal even code, for a given set of symbols, each one with a given probability. The algorithm employs dynamic programming and its complexity is $O(n^3)$, where $n$ is the number of symbols. Next, we propose two heuristics for approximating an optimal code, based on Huffman’s algorithm. The time required for computing an even code using both heuristics is $O(n \log n)$. We show that the cost of an even code constructed by the second heuristic is at most 16.7% higher than the cost of a Huffman code for the same probabilities. That is, less than 16.7% higher than the corresponding optimal even code. However, for practical purposes, this value seems to be much less. In fact, we have performed several computer experiments, obtaining values always less than 5%, except for small $n$. This number corresponds to the overhead in the size of the encoded message, for having the ability to detect an odd number of 1-bit errors.

The plan of the paper is as follows. In Section 2 we describe the exact algorithm for constructing an optimal even code. The heuristics are formulated in Section 3. In Section 4 we present bounds for the cost difference between even trees and corresponding Huffman trees. A probabilistic model to evaluate the error detection capability of an even tree is developed in Section 5. In Section 6 we present experimental results about costs and error detection probabilities in even trees.

The following definitions are of interest.

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of elements, called symbols. Each $s_i \in \mathcal{S}$ has an associated probability $f_i$.

Throughout the paper, we assume $f_i \leq f_{i+1}$, for $1 \leq i < n$. We generally consider that the sum of probabilities $f_1, \ldots, f_n$ to be 1, although this fact is not necessary for optimality considerations.
An encoding $e_i$ for a symbol $s_i \in \mathcal{S}$ is a finite sequence of 0's and 1's, associated to $s_i$. Each 0 and 1 is a bit of $e_i$. The parity of $e_i$ is the parity of the number of 1's contained in $e_i$. A subsequence of $e_i$ starting from its first bit is a prefix of $e_i$. The set of encodings for all symbols of $\mathcal{S}$ is a code $\mathcal{C}$ for $\mathcal{S}$. A code in which every encoding does not coincide with a prefix of any other encoding is a prefix code.

A message $M$ is a sequence of symbols. The encoded message of $M$ is the corresponding sequence of encodings. The parity of an encoded message is the number of 1's contained in it.

A binary tree is a rooted tree $T$ in which every node $z$, other than the root, is labeled left or right in such a way that any two siblings have different labels. Say that $T$ is trivial when it consists of a single node. A binary forest is a set of binary trees. A path of $T$ is a sequence of nodes $z_1, \ldots, z_t$, such that $z_i$ is the parent of $z_{i+1}$, for $1 \leq i < t$. The value $t - 1$ is the size of the path, whereas all $z_i$ are descendants of $z_1$. If $z_1$ is the root then $z_1, \ldots, z_t$ is a root path and, in addition, if $z_t$ is a leaf, then $z_1, \ldots, z_t$ is a root-leaf path of $T$. The depth of a node is the size of the root path to it. For a node $z$ of $T$, $T(z)$ denotes the subtree of $T$ rooted at $z$, that is, the binary tree containing all descendants of $z$ in $T$ (including $z$ itself). The left subtree of $z$ is the subtree $T(z')$, where $z'$ is the left child of $z$. Similarly, define the right subtree of $z$. The left and right subtrees of the root of $T$ are denoted by $T_L$ and $T_R$, respectively. A strictly binary tree is one in which every node is a leaf or has two children. A full binary tree is a strictly binary tree in which all root-leaf paths have the same size. A complete binary tree is a binary tree where the null subtrees are located at the last two levels. In a binary tree $T$, the edges of $T$ leading to left children are labeled 0, whereas those leading to right children are labeled 1. The parity of a node $z$ is the parity of the number of 1's among the edges forming the root path to $z$. A node is even or odd, according to its parity, respectively.

A (binary tree) representation of a code $\mathcal{C}$ is a binary tree $T$ such that there exists a one-to-one correspondence between encodings $e_i \in \mathcal{C}$ and root-leaf paths $p_i$ of $T$ in such a way that $e_i$ is precisely the sequence of labels, 0 or 1, of the edges forming $p_i$. A code admits a binary tree representation if and only if it is a prefix code. Let $d_i$ be the depth of the leaf of $T$ associated to the symbol $s_i$. Define the cost of $T$ as the sum $c(T) = \sum_{i=1}^{n} f_i d_i$, Hence, the cost of a trivial tree is 0. An optimal code (tree) is one with the least cost. A full representation tree of $\mathcal{C}$ is a binary tree $T^*$ obtained from the representation tree $T$ of $\mathcal{C}$, by adding a new leaf as the second child of every node having exactly one child. The original leaves of $T$ are the encoding leaves, whereas the newly introduced leaves are the error leaves. Clearly, in the case of Huffman trees, there are no error leaves.

An even (odd) code is a prefix code in which all encodings are even (odd). Similarly, an even (odd) tree is a tree representation of an even (odd) code.
Examples of even trees for up to three symbols appear in Figure 2, while Figure 3 depicts an optimal even tree for 11 symbols with uniform probabilities.

It is easy to see that even codes detect the occurrence of an odd number of 1-bit errors in a message, as follows. We know that all the encodings are even, so the encoded message is also even. By introducing an odd number of errors, the encoded message becomes odd. Since the encodings are even, the latter implies that in the full tree representation of the code, an error leaf would be hit during the decoding process, or otherwise the process terminates at some odd node of the tree. It should be noted that odd codes do not have this property. For example, if we have a code $\mathcal{C} = \{1, 01\}$ and a message 01, if the first bit is changed, resulting 11, the message would be wrongly decoded without signaling error.

2 Exact Algorithm

In this section, we describe an exact algorithm for constructing an optimal even tree for symbols with arbitrary probabilities.
Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of symbols, each $s_i$ having probability $f_i$ satisfying $f_i \leq f_{i+1}$, for $1 \leq i < n$. For $m \leq n$, denote $\mathcal{S}_m = \{s_1, \ldots, s_m\}$. Our aim is to find an even code $\mathcal{C}$ for $\mathcal{S}$ having minimum cost. In fact, we propose a solution for a slightly more general problem.

A parity forest $F$ for $\mathcal{S}_m$ is a set of $q$ even trees and $q$ odd trees, for some $q \in \{1, \ldots, m\}$, such that the even parity leaves of even trees and odd parity leaves of odd trees correspond to the symbols of $\mathcal{S}_m$. Define the cost of $F$ as the sum of the costs of its trees. Say that $F$ is $(m, q)$-optimal when its cost is the least among all forests for $\mathcal{S}_m$ having $q$ even trees and $q$ odd trees. Denote by $c(m, q)$ the cost of an $(m, q)$-optimal forest. First, define the function

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

In terms of this notation, the solution of our problem is a tree having as subtrees the trees of an $(n, 1)$-optimal forest. Its cost is $c(n, 1) + A_n$.

The following theorem describes the computation of $c(m, q)$.

**Theorem 1** Let $m, q$ be integers such that $1 \leq m \leq n$ and $m \geq q \geq 1$.

(1) if $m \leq q$ then $c(m, q) = 0$

(2) if $m > q$, then $c(m, q) = \min_{0 \leq i \leq q} \{c(m - i, 2q - i) + A_{m-i}\}$

**Proof.** By induction, we show that cases (1)-(2) correctly compute $c(m, q)$, for $1 \leq m \leq n$ and $m \geq q \geq 1$. When $m = 1$, case (1) implies that $c(1, q) = 0$, which is correct since the only symbol is in the root of an even tree. We have also one odd root as an error leaf and the remaining trees are null. For $m > 1$, let $F$ be an $(m, q)$-optimal forest for $\mathcal{S}_m$. Consider the following alternatives.

(1) $m \leq q$: In this case, $m$ trivial even trees of $F$ correspond to the symbols $s_1, \ldots, s_m$, respectively. The remaining $q - m$ even trees are null trees. We also have $m$ trivial odd trees corresponding to error leaves and $q - m$ null odd trees. Then $c(m, q) = 0$.

(2) $m > q$: In this case, we can have $i$ trivial even trees of $F$ corresponding to the symbols $s_{m-i+1}, \ldots, s_m$, for some $i \in \{0, \ldots, q\}$. The remaining $m - i$ symbols are distributed in a forest $F'$ with $q - i$ even trees and $q$ odd trees. Note that the forest $F''$ formed by removing the roots of the trees belonging to $F'$ is an $(m - i, 2q - i)$-optimal forest. This implies that the cost of $F$ is given by $c(m - i, 2q - i) + A_{m-i}$. Therefore, $c(m, q)$ is the minimum of $c(m - i, 2q - i) + A_{m-i}, 0 \leq i \leq q$. 


Theorem 1 leads to a dynamic programming algorithm for determining $c(m, q)$, for all $1 \leq m \leq n$ and $m \geq q \geq 1$.

Start by evaluating the function $A_i$ for $1 \leq i \leq n$. The parameter $m$ varies increasingly, $1 \leq m \leq n$. The first cost to be computed is $c(m, m)$, which is 0, by (1). For each such $m$, vary $q$ decreasingly, $1 \leq q < m$, and for each such pair $m, q$, compute $c(m, q)$ applying (2). The computation stops when $c(n, 1)$ is calculated, since our target is to obtain $c(n, 1) + A_n$. There are $O(n^2)$ subproblems. The evaluation of each one is performed in constant time, if by equation (1), or in $O(n)$ time, when the evaluation is by (2). Consequently, the time complexity is $O(n^3)$. The space requirements are $O(n^2)$.

3 Heuristics

In this section we describe two heuristics to obtain even codes. Heuristics 1 is very simple and is based on a slight modification of the classical Huffman algorithm [1]. Heuristics 2 adds possible improvements to the previous one. As we shall see, these improvements allow to yield even codes very close to the optimal ones.

3.1 Heuristics 1

Given $n$ symbols with probabilities $f_1, f_2, \ldots, f_n$, Heuristics 1 consists of two steps:

Step 1. Run Huffman’s algorithm in order to obtain a Huffman tree $T_H$ for the $n$ symbols.

Step 2. Convert $T_H$ into an even tree $T_E$, in the following way: for each odd leaf $z$ corresponding to a symbol $s_i$, create two children $z_L$ and $z_R$ such that:
- the left child $z_L$ is an error leaf;
- the right child $z_R$ is the new encoding leaf corresponding to $s_i$. We call $z_R$ an augmented leaf.

Observe that the overall running time of Heuristics 1 is $O(n \log n)$, since it is dominated by Step 1. Step 2 can be easily done in $O(n)$ time.

3.2 Heuristics 2

Now we present three possible ways to improve the heuristics previously described. As we shall see, these improvements do not increase the running time.
in practice, while producing a qualitative increase of performance with respect to the cost of the generated code.

**Improvement I.** During Step 1 (execution of Huffman’s algorithm), add the following test:

*Among the candidate pairs of the partial trees to be merged at the beginning of a new iteration, break ties by giving preference to a pair of trees $T_1$ and $T_2$ such that $T_1$ is trivial and $T_2$ is not.*

In other words, the idea is to avoid merging trivial trees as much as possible. The reason why this strategy is employed is explained in the sequel.

In $T_{H_1}$ there exist two sibling leaves for each merge operation of trivial trees occurring along the algorithm. Of course, one of the siblings is guaranteed to be an odd leaf. When we force a trivial tree to be merged with a non-trivial one, we minimize the number of pairs of sibling leaves in $T_{H_1}$, and thus the number of those “guaranteed odd” leaves. In many cases, this strategy lowers the additional cost needed to produce the even tree in Step 2.

Let us denote by $T_{H_1}$ the Huffman tree obtained by Improvement I. It is worth remarking that this improvement does not affect the essence of Huffman’s algorithm, since $T_{H_1}$ is a plausible tree.

Moreover, it is possible to implement Improvement I in constant time by keeping two heaps $H'$ and $H''$ during the execution of Huffman’s algorithm, where the nodes of $H'$ contain trivial trees and the nodes of $H''$ the remaining ones. At the beginning of the algorithm, $H'$ contains $n$ nodes and $H''$ is empty. When starting a new iteration, simply test whether the roots of $H'$ and $H''$ form a candidate pair of partial trees to be merged; if so, merge them.

**Improvement II.** Change $T_{H_1}$ by repeatedly applying the following operation in increasing depth order:

*If there exist two nodes $z', z''$ at the same depth of $T_{H_1}$ such that $z'$ is an odd leaf and $z''$ is an even internal node, exchange the positions of $z'$ and $z''$.*

Observe each single application of the above operation decreases the number of odd leaves in $T_{H_1}$ by one unit. Each time we find $k$ odd leaves and $\ell$ even internal nodes at some depth $i$, we perform $\min\{k, \ell\}$ changes and proceed to depth $i + 1$.

It is clear that the number of changes is bounded by the number of leaves of $T_{H_1}$. Since a single change can be done by modifying a constant number of pointers, the overall complexity of Improvement II is $O(n)$.

Denote by $T_{H_2}$ the Huffman tree obtained by Improvement II. Again, the essence
of Huffman’s algorithm is not affected, since $T_{H_2}$ is still plausible.

**Improvement III.** Apply Step 2 on $T_{H_2}$. Let $T$ be the even tree obtained. Then redistribute the symbols among the leaves of $T$ as follows:

Whenever there exist two leaves $z_i, z_j$ (of even parities) in $T$ with depths $d_i \leq d_j$, representing symbols $s_i, s_j$ with probabilities $f_i \leq f_j$, respectively, then exchange the symbols assigned to $z_i$ and $z_j$.

Observe that each single re-assignment performed above reduces the cost of the resulting even tree by $(d_j - d_i)(f_j - f_i)$.

The entire process can be implemented in the following way: after applying Step 2, order the leaves $z_1, z_2, \ldots, z_n$ of $T$ according to its respective depths $d_1, d_2, \ldots, d_n$ using bucket sort. Then reassign the leaves to symbols, such that leaf $z_i$ with depth $d_i$ is assigned to the symbol $s_{n-i+1}$ with probability $f_{n-i+1}$. (recall that $f_1 \leq f_2 \leq \ldots f_n$). With this procedure we restore distortions possibly introduced by Improvement II. The time required for this operation is therefore $O(n)$. Consequently, the overall time bound for Heuristics 2 is $O(n \log n)$, with $O(n)$ space.

Other possible improvements were initially considered and then discarded, as they did not bring experimental relevant advantages. One of those marginal improvements is to exchange an error leaf with a subtree whose root is deeper than that error leaf.

### 4 Bounds

In this section, we present an analytical upper bound for the cost of the even tree generated by Heuristics 2 with respect to the cost of the corresponding Huffman tree.

The terminology employed in this section is the following: given $n$ symbols with probabilities $f_1, f_2, \ldots, f_n$, $T_H$ is the Huffman tree for these symbols; $T_E$ is the corresponding optimal even tree; $T_U$ is the even tree obtained by applying Heuristics 1; and $T_{U_2}$ is the even tree obtained by applying Heuristics 2. Observe that

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

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

**Proof.** This bound is due to Improvements II and III. Let $n_o(k), n_e(k), n_{o1}(k), n_{e1}(k)$ be, respectively, the number of odd leaves, odd internal nodes, even leaves
and even internal nodes at depth $k$ of $T_{H_2}$. Then either $n_{d}(k) = 0$ or $n_{e}(k) = 0$ (1). Moreover, it is clear that $n_{d}(k) + n_{e}(k) = n_{d}(k) + n_{e}(k)$ (2).

We claim that $n_{d}(k) \leq n_{d}(k)$. Otherwise, if $n_{d}(k) > n_{d}(k)$, then $n_{d}(k) > 0$, which implies $n_{e}(k) = 0$ from (1). But in this case $n_{d}(k) + n_{e}(k) = n_{d}(k)$ from (2), that is, $n_{d}(k) \leq n_{d}(k)$, a contradiction.

By summing up $n_{d}(k)$ and $n_{d}(k)$ for all values of $k$, we conclude that the number of odd leaves is less than or equal to the number of even leaves in $T_{H_2}$. That is, the number of odd leaves is at most $\lfloor \frac{n}{4} \rfloor$ and is less than or equal to the number of merge operations between two trivial trees in Huffman’s algorithm. Next, Step 2 puts odd leaves one level deeper, in order to convert $T_{H_2}$ into an even tree.

Now, when applying Improvement III, the probabilities are redistributed in the tree, in such a way that two leaves $z_i, z_j$ with depths $d_i, d_j$ and probabilities $f_i, f_j$ satisfy the condition $f_i \geq f_j \Leftrightarrow d_i \leq d_j$. Consequently, there exists a one-to-one correspondence between the set of augmented leaves and a subset of the even leaves such that if $z_i$ is an augmented leaf and $z_j$ is its corresponding even leaf then $f_i \geq f_j$. Thus:

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

Lemma 2 states that the greatest difference in cost between an even tree obtained by Heuristics 2 and the corresponding Huffman tree is at most 0.5. Next theorem gives another bound that can be tighter for low entropy distributions, if the cost of the Huffman tree is lower than 3.

**Theorem 3** $c(T_{U_2}) \leq \frac{7}{5} c(T_{H})$, if $n > 4$.

![Huffman trees for $n = 5$](image)

**Proof.** In this proof $T_{H}$ is a Huffman tree created by the process defined by Heuristics 2, and $T_{U_2}$ the corresponding even tree. Suppose, initially, $n = 5$. 

10
The only three possible forms for $T_H$ are depicted in Figure 4. Let $f_1 \leq f_2 \leq f_3 \leq f_4 \leq f_5$ be the probabilities of the leaves $l_1, \ldots, l_5$, respectively. Consider each one of those trees.

**Case 1.** Let $T_H$ be the top left Huffman tree in Figure 4. Then $T_{V_2}$ is created from $T_H$ by moving leaf $l_1$ one level down and consequently turning its parity even. Thus:

$$c(T_{V_2}) = 5f_1 + 4f_2 + 3f_3 + 2f_4 + f_5$$

$$= c(T_H) + f_1$$

$$= (1 + \frac{f_1}{c(T_H)})c(T_H)$$

$$= (1 + \frac{f_1 + f_2 + f_3 + 2f_4 + f_5}{c(T_H)})c(T_H)$$

$$\leq (1 + \frac{f_1 + f_2}{c(T_H)})c(T_H) = \frac{7}{6}c(T_H).$$

**Case 2.** Now let $T_H$ be the right top tree in Figure 4. Then $T_{V_2}$ is created from $T_H$ by moving leaves $l_1$ and $l_4$ one level down, turning their parities even, and interchanging leaves $l_2$ and $l_4$. Thus:

$$c(T_{V_2}) = 4f_1 + 4f_2 + 3f_3 + 3f_4 + f_5$$

$$= c(T_H) + f_1 + f_2$$

$$= (1 + \frac{f_1 + f_2}{c(T_H)})c(T_H)$$

$$= (1 + \frac{f_1 + f_2}{3f_1 + 3f_2 + 3f_3 + 3f_4 + f_5})c(T_H)$$

$$\leq (1 + \frac{f_1 + f_2}{6(f_1 + f_2) + f_5})c(T_H)$$

$$\leq (1 + \frac{f_1 + f_2}{6(f_1 + f_2)})c(T_H)$$

$$\leq \frac{7}{6}c(T_H).$$

**Case 3.** Finally, let $T_H$ be the bottom tree in Figure 4. Then $T_{V_2}$ is created from $T_H$ by moving leaves $l_1$ and $l_4$ one level down, turning their parities even, and interchanging leaves $l_4$ and $l_5$. Thus:

$$c(T_{V_2}) = 4f_1 + 3f_2 + 3f_3 + 2f_4 + 2f_5$$

$$= c(T_H) + f_1$$

$$= (1 + \frac{f_1}{c(T_H)})c(T_H)$$

$$= (1 + \frac{f_1 + f_2 + f_3 + f_4 + f_5}{c(T_H)})c(T_H)$$

$$\leq (1 + \frac{f_1 + f_2}{c(T_H)})c(T_H) = \frac{7}{6}c(T_H).$$

11
\[ = c(T_H) + f_1 + f_3 \]

\[ = (1 + \frac{(\frac{1}{c(T_H)} )c(T_H)}{c(T_H)} )c(T_H) \]

\[ = (1 + \frac{f_1 + f_3}{c(T_H)} )c(T_H) \]

\[ \leq (1 + \frac{f_1 + f_3}{\sigma d(T_H)} )c(T_H) \]

\[ \leq \frac{\sigma}{\sigma} c(T_H). \]

Suppose now \( n > 5 \). Note that leaf \( l_n \) is the nearest leaf to the root. We consider three possibilities for its depth \( d_n \): \( d_n = 1 \), \( d_n = 2 \) and \( d_n > 2 \), and we show that, in each case, the hypothesis remains valid.

**Case a.** \( d_n = 1 \).

In this case, \( T_H \) is the tree illustrated in Figure 5. The subtree \( T_{H_21} \) has \( n_1 \geq 5 \) leaves. \( T_{H_21} \) has the same form of \( T_H \). Let \( T_{H_21} \) be the subtree of \( T_{H_21} \) corresponding to \( T_{H_21} \). The result can be proved by induction.

Since the number of leaves of \( T_{H_21} \) is greater than 4, suppose we have \( c(T_{H_21}) \leq \frac{\sigma}{\sigma} c(T_{H_21}) \). Once \( c(T_{H_21}) = 1 + c(T_{H_21}) \), it follows that:

\[ c(T_{H_21}) \leq 1 + \frac{\sigma}{\sigma} c(T_{H_21}) \leq \frac{\sigma}{\sigma}(1 + c(T_{H_21})) = \frac{\sigma}{\sigma} c(T_H). \]

Consequently, the result is also valid for \( T_H \) and \( T_{H_21} \).

---

**Figure 5:** Huffman tree for \( n > 5 \), \( d_n = 1 \)

**Figure 6:** Huffman tree for \( n > 5 \), \( d_n = 2 \)
Case b. \( d_n = 2 \).

In this case, \( T_H \) can be considered as the tree illustrated in Figure 6, where \( l_n \) is a leaf and \( T_{H_{21}}, T_{H_{22}} \) and \( T_{H_{23}} \) are disjoint subtrees having \( n_1, n_2 \) and \( n_3 \) leaves, respectively. Let \( x, y \) and \( z \) be the sums of the probabilities of the leaves of subtrees \( T_{H_{21}}, T_{H_{22}} \) and \( T_{H_{23}} \), respectively. Clearly, \( f_n + x + y + z = 1 \). We must consider 2 subcases:

Subcase b.1 \( T_{H_{21}} \) is not a leaf, \( T_{H_{22}} \) is an odd leaf, and \( T_{H_{23}} \) is an even leaf.

In this case, \( T_{U_2} \) is obtained from \( T_H \) by moving \( T_{H_{22}} \) one level down and modifying \( T_{H_{23}} \) with the same rules of Heuristics 2, in order to obtain \( T_{U_2} \). That is,

\[
c(T_{U_2}) = 2 \cdot (f_n + x + y + z) + c(T_{U_2}) + y = 2 + c(T_{U_2}) + y.
\]

By Lemma 2, \( c(T_{U_2}) \leq c(T_{H_{21}}) + x/2 \). Since \( c(T_H) = 2 + c(T_{H_{21}}) \), it follows that:

\[
c(T_{U_2}) \leq 2 + c(T_{H_{21}}) + x/2 + y = (1 + \frac{x/2+y}{2c(T_{H_{21}})}) c(T_H).
\]

Since \( c(T_{H_{21}}) \geq x \), we have \( c(T_{U_2}) \leq (1 + \frac{x/2+y}{x}) c(T_H) = (\frac{3}{2} + \frac{y}{x}) c(T_H) \).

The fraction \( \frac{y}{x} \) attains its maximum value when \( y = z = f_n \). This implies \( x = 1 - 3y \), and, consequently:

\[
c(T_{U_2}) \leq (\frac{3}{2} + \frac{1}{1-3y}) c(T_H) = \frac{3}{2} c(T_H).
\]

Subcase b.2. In this case, \( T_{H_{21}} \) and \( T_{H_{22}} \) are not leaves. \( T_{H_{23}} \) may be a single leaf or a subtree with more than one leaf.

\( T_{U_2} \) is obtained from \( T_H \) by applying Heuristics 2 to subtrees \( T_{H_{21}}, T_{H_{22}} \) and \( T_{H_{23}} \) to obtain \( T_{U_2}, T_{U_2}, \) and \( T_{U_2} \), respectively. Considering that leaf \( l_n \) is not changed by the heuristics, by Lemma 2 we have:

\[
c(T_{U_2}) \leq c(T_H) + \sum_{i=1}^{n-1} f_i/2
\]

\[
= c(T_H) + (x + y + z)/2
\]

\[
= (1 + \frac{x+y+z}{2c(T_H)}) c(T_H)
\]

\[
= (1 + \frac{x+y+z}{2(2+f_n+c(T_{H_{21}})+c(T_{H_{22}})+c(T_{H_{23}}))}) c(T_H)
\]

Since \( c(T_{H_{21}}) + c(T_{H_{22}}) + c(T_{H_{23}}) \geq x + y + z < 1 \), we have:

\[
c(T_{U_2}) \leq (1 + \frac{x+y+z}{2(2+f_n+x+y+z)}) c(T_H)
\]
\[ = (1 + \frac{1}{2(n+1)}) c(T_H) \]
\[ = (1 + \frac{1}{6}) c(T_H) \]
\[ = \frac{7}{6} c(T_H). \]

**Case c.** \( d_n > 2. \)

In this case, we have \( c(T_H) \geq 3 \) and \( c(T_{V_2}) \leq c(T_H) + 1/2. \) Consequently:

\[ c(T_{V_2}) \leq (1 + \frac{1}{2(n+1)}) c(T_H) \leq (1 + \frac{1}{6}) c(T_H) = \frac{7}{6} c(T_H). \]

In all cases, \( c(T_{V_2}) \leq \frac{7}{6} c(T_H). \)

---

The bound given by Theorem 3 cannot be made better. We show this fact by exhibiting an infinite family \( F \) of Huffman trees where, given the approximation \( \epsilon > 0, \) we always can find a tree \( T_H \in F \) such that, for the corresponding tree \( T_{V_2} \) obtained from Heuristics 2, we have \( \frac{7}{6} c(T_H) - c(T_{V_2}) \leq \epsilon. \)

This family is illustrated in Figure 7. Each tree of the family is characterized by two parameters \( n \) and \( q, \) \( n \geq 4 \) and \( \frac{1}{n} \geq q > 0. \) Each tree has \( n \) leaves and depth \( n - 2. \) Three leaves \( l_{n-2}, l_{n-1}, \) and \( l_n \) have depth 2 and probability \( \frac{1}{3}. \)

The remaining \( n - 3 \) leaves \( l_1, \ldots, l_{n-3} \) are arranged in the tree as follows: one leaf \( l_1 \) has depth \( n - 2 \) and probability \( \frac{1}{n-1}; \) \( n - 4 \) leaves \( l_i, \) \( 2 \leq i \leq n - 3 \) have depths \( n - i \) and probabilities \( \frac{1}{n(n-1)} \), respectively. In this family of trees, there are only 2 odd parity leaves: \( l_1 \) and \( l_{n-1}. \)

**Theorem 4** Let \( T_H \) be a tree belonging to \( F. \) Then \( T_H \) is a Huffman tree.
**Proof.** Let $A_j = \sum_{i=1}^{j} f_i$. Note that $A_{n-3}$ satisfies:

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

Thus, the overall sum of probabilities in $T_H$ is $3\frac{1-2^2}{3} + q = 1$.

Next, a necessary condition for $T_H$ to be a Huffman tree is that $A_{n-3}$ should not be greater than $f_{n-1} + f_n$. Hence, we must have $q \leq 2 \frac{1-2^2}{3}$, which implies that $q \leq \frac{2}{5}$.

In addition, let us show that $A_{i-1} = f_i$, for $1 < i \leq n - 3$:

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

Consequently, the leaf $l_i$ is joined correctly to the subtree containing leaves $l_1, \ldots, l_{i-1}$ during the process of creation of $T_H$. Hence, $T_H$ is indeed a Huffman tree. \(\blacksquare\)

**Theorem 5** Given $\epsilon > 0$, there exists $T_H \in F$ such that the corresponding even tree $T_{\nu_2}$ obtained from $T_H$ by applying Heuristics 2 satisfies $\frac{7}{10}c(T_H) - c(T_{\nu_2}) \leq \epsilon$.

**Proof.** Let $T_H \in F$. Then:

$$c(T_H) = (n - 2)\frac{q}{2^{n-4}} + \sum_{i=2}^{n-3} (n - i)\frac{q}{2^{n-i-2}} + 3.2\frac{1-q}{3}$$

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

Now, observe that leaves $l_2, \ldots, l_{n-3}$ of $T_H$ have even parity, and leaves $l_1$ and $l_{n-1}$ have odd parity (see Figure 7). Then, the only difference between the even tree $T_{\nu_2}$ obtained from $T_H$ after applying Heuristics 2 is that leaves $l_1$ and $l_{n-1}$ of $T_{\nu_2}$ are shifted one level below. Therefore:

$$c(T_{\nu_2}) = c(T_H) + \frac{q}{2^{n-4}} + \frac{1-q}{3} = c(T_H) + \frac{3q + 2^{n-4}(1-q)}{3^{2n-4}}.$$ 

15
Given $\epsilon > 0$, we want to choose $q$ such that 
\[ \frac{2^n(1 + q) - 4q}{6 \cdot 2^{n-3}} - \frac{3q + 2^{n-4}(1 - q)}{3 \cdot 2^{n-4}} \leq \epsilon \]
which is equivalent to
\[ \frac{q(2^{n-5} - 1)}{3 \cdot 2^{n-5}} \leq \epsilon \]
Since $\frac{(2^{n-5} - 1)}{2^{n-5}} < 1$, choose $q \leq \min\left(\frac{2}{3}, 3\epsilon\right)$. 

We found an exact upper bound for the cost difference between a Huffman tree and an even tree obtained by Heuristics 2. Clearly this is also an upper bound (possibly not exact) for the cost difference between an optimal even tree and the corresponding Huffman tree. Nevertheless, it remains open the determination of a tight upper bound for this difference. Experimental results suggest that it is quite lower than the bound given above.

5 Probabilistic Model for Error Detection

In this section we develop a probabilistic model to assess the error detection capability of an even tree. In the next section we present some experimental results to validate the model.

Let us suppose that one or more errors occurred in a coded message, that is, one or more bits were changed. After the first error, we can have a sequence of wrong decoded symbols. The following model evaluates the probability of a sequence of $k$ wrong decodings and the overall probability to catch the error occurred.

Let $s_i$ be the first symbol in a coded message where the error was introduced and $T$ the corresponding even tree used to encode the message. First we shall calculate the probability $P_p(T, k)$ of the next $k$ symbols (including $s_i$) to be wrongly decoded. $P_p(T, k)$ is the probability of error propagation through the next $k - 1$ symbols.

After the introduction of the first error, the remaining bit string to be decoded can be considered as a random string. In terms of the decoding tree, this means that each deviation in the tree has the same probability, which is $1/2$. Consider that the first wrong bit corresponds to the root of the tree. Then, the probability
of leaf $l_j$ be hit is $P_l(j) = 2^{-d_j}$, where $d_j$ is the depth of $l_j$. We define two parameters for $T$: $Pt_e(T)$ and $Pt_e(T)$. $Pt_e(T)$ is the probability of any encoding leaf be reached starting from the root, and its value is $Pt_e(T) = \sum_{j=1}^{n} 2^{-d_j}$. $Pt_e(T)$ is the corresponding probability, considering error leaves and is given by $Pt_e(T) = \sum_{j=1}^{N_e(n)} 2^{-d_j}$, where $N_e(n)$ is the number of error leaves of $T$.

Clearly, $Pt(T) + Pt_e(T) = 1$.

The error propagation probability through $k$ symbols is then $P_p(T, k) = Pt_e(T)^k = (1 - Pt_e(T))^k$, decreasing exponentially with $k$. Note that to consider that the first error had occurred exactly in the root only reduces this probability, once error leaves are placed in the two last levels of the tree. So, we will neglect the fact that the error can also occur in any node other than the root.

Now let us consider how to evaluate the error detection capability of $T$. Let $b$ be the average number of symbols in a message. The error detection probability of $T$ for messages with $b$ symbols, $P_m(T, b)$, is the complement of the probability that the errors propagate until the end of the message.

If errors in the message are independent and one error occurs with probability $q$, then the probability that the bit $i$ of the message is the first wrong bit is given by

$$\sum_{k=1}^{b-i+1} q^k \binom{b-i}{k} \binom{b}{k},$$

which can be approximated to $q/b$ if $q$ is small. That is, this probability is uniform along the $b$ symbols in the message, and its value is $1/b$. Then $P_m(T, b)$ can be estimated as:

$$P_m(T, b) = 1 - \sum_{k=1}^{b} P_p(T, k)/b = 1 - (1 - Pt_e(T) - (1 - Pt_e(T))^{b+1})/(b, Pt_e(T)).$$

We can refine this model, by considering the number of errors introduced in the message and not only the first error. Once $T$ detects all odd number of wrong bits, and considering that half of all errors situations are related to an odd number of wrong bits, we divide the probability of error propagation by 2. We can yet consider that, in the decoding procedure of the last symbol, the process will not necessarily finish in a leaf. It may be the case that the last node hit is an internal node. If the procedure ends at an internal node, then the error is pointed by the decoding process. As the number of internal nodes is not less than the number of encoding leaves, we will again divide the error propagation probability by 2.
With this refinement, we have a new estimation:

\[ P_m(T, b) = 1 - \sum_{k=1}^{b} \left( 1 - Pt_e(T) \right)^{b-1} / 4b \]

\[ = 1 - (1 - (1 - Pt_e(T))^{b}) / (4b Pt_e(T)) \]

The above expression shows that, to estimate the error detection capability of an even tree \( T \), the main parameter to consider is \( Pt_e(T) \), related to error leaves. In the sequel, we explore the bounds for this parameter, given by the next two theorems.

**Theorem 6** If \( T \) is an optimal even tree for \( n \) symbols having uniform probabilities then \( 1/16 \leq Pt_e(T) \leq 1/4 \).

**Proof.** By definition, we have:

\[ Pt_e(T) = \sum_{i=1}^{N_e(T)} 2^{-d_i} \]

where \( N_e(T) \) is the number of error leaves in \( T \) and \( d_i \) is the depth of the error leaf \( l_i \).

Observe that, in \( T \), for each error leaf, there are two associated encoding leaves: its sibling and the sibling of its parent. Thus, considering these encoding leaves, we have:

\[ Pt_e(T) \geq \sum_{i=1}^{N_e(T)} (2^{-d_i+1} + 2^{-d_i}) = 3Pt_e(T). \]

Since \( Pt_e(T) + Pt_e(T) = 1 \), we have \( Pt_e(T) + 3Pt_e(T) \leq 1 \). Thus,

\[ Pt_e(T) \leq 1/4. \]

To prove the lower bound \( 1/16 \), we can use the fact that, for \( n \geq 5 \), an optimal even tree can be thought as the composition of basic optimal even/odd trees having two, three or four encoding leaves, and one error leaf, as depicted in Figure 8. For instance, the optimal tree for \( n = 11 \) depicted in Figure 3 is composed as follows: the subtrees of the root must have five and six nodes; the left subtree with five nodes, by its turn, is composed by \( T_3 \) and a tree isomorphic to \( T_3 \); and the right subtree is composed by \( T_3 \) and another tree isomorphic to
Figure 8: Optimal even trees $T_2$, $T_3$ and $T_4$, for $n = 2, 3$ and 4, respectively.

$T_3$. Hence, this process can be recursively performed in a top-down fashion for every $n \geq 3$.

Among those basic three trees, the tree for $n = 4$ has the lowest relation $ne/n$ where $ne$ is the number of error leaves and $n$, the number of encoding leaves. The even tree which is composed only with that basic structure has minimum $Pt_e(T)$. It occurs when the number of symbols, $n$, is of the form $n = 2^k$, $k > 1$ integer. In this situation, the tree has $2^{k-2}$ error leaves, all having depth equal to $k + 2$, as it is presented in [12]. Thus,

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

Considering the two bounds, $1/16 \leq Pt_e(T) \leq 1/4$. 

**Theorem 7** If $T$ is an optimal even tree for $n$ symbols, having arbitrary probabilities, then $2^{-n} \leq Pt_e(T) \leq 1/4$.

**Proof.** The upper bound $1/4$ is obtained exactly as in the previous theorem. Now, with respect to the lower bound, note that the optimal even tree $T$ having minimum $Pt_e(T)$ is a special one, having depth $= n$. At each level $i \in \{2, \ldots, n + 1\}$, there is one encoding leaf having even parity. Consequently, there is only one error leaf at the last level. Thus, $Pt_e(T) = 2^{-n}$.

Concluding, for arbitrary probabilities we have:

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

The bounds of Theorems 6 and 7 are given for optimal even trees. If we consider even trees obtained by Heuristics 2, the only difference we have is the lower
bound for uniform probabilities. Heuristics 2 always uses trees isomorphic to $T_2$ and $T_3$ (see Figure 8). If $T$ is an even tree obtained by Heuristics 2 with minimum $Pt_e(T)$, then $T$ must be composed only by trees isomorphic to $T_3$. This situation occurs for $n$ of the form $n = 3.2^k$, $k \geq 0$. Hence, $T$ has $2^k$ error leaves, all having depth equal to $k + 3$, as it is presented in [12]. Thus,

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

Considering the two bounds, $1/8 \leq Pt_e(T) \leq 1/4$.

6 Experimental results

In this section we present two kinds of experimental results: those related to costs of even trees, and those related to error detection in even trees.

6.1 Experimental Results on Costs

First we consider experimental results related to costs, summarized in Tables 2 to 4. The tables present the costs of the trees obtained by the algorithms, for several values of $n$. They were obtained via a program written in Pascal, running on a Pentium IV computer with 1.8 GHz and 256M RAM.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$c(T_H)$ (A)</th>
<th>$c(T_E)$ (B)</th>
<th>$c(T_{v_1})$ (C)</th>
<th>$c(T_{v_2})$ (D)</th>
<th>$\frac{B-A}{A}$</th>
<th>$\frac{C-A}{A}$</th>
<th>$\frac{D-A}{A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>6.00</td>
<td>6.50</td>
<td>6.50</td>
<td>6.50</td>
<td>8.3</td>
<td>8.3</td>
<td>8.3</td>
</tr>
<tr>
<td>128</td>
<td>7.00</td>
<td>7.50</td>
<td>7.50</td>
<td>7.50</td>
<td>7.1</td>
<td>7.1</td>
<td>7.1</td>
</tr>
<tr>
<td>192</td>
<td>7.67</td>
<td>8.00</td>
<td>8.17</td>
<td>8.00</td>
<td>4.4</td>
<td>6.5</td>
<td>4.4</td>
</tr>
<tr>
<td>256</td>
<td>8.00</td>
<td>8.50</td>
<td>8.50</td>
<td>8.50</td>
<td>6.3</td>
<td>6.3</td>
<td>6.3</td>
</tr>
<tr>
<td>320</td>
<td>8.40</td>
<td>8.80</td>
<td>8.90</td>
<td>8.80</td>
<td>4.8</td>
<td>6.0</td>
<td>4.8</td>
</tr>
<tr>
<td>384</td>
<td>8.67</td>
<td>9.00</td>
<td>9.17</td>
<td>9.00</td>
<td>3.9</td>
<td>5.8</td>
<td>3.9</td>
</tr>
<tr>
<td>448</td>
<td>8.86</td>
<td>9.29</td>
<td>9.36</td>
<td>9.29</td>
<td>4.8</td>
<td>5.7</td>
<td>4.8</td>
</tr>
<tr>
<td>512</td>
<td>9.00</td>
<td>9.50</td>
<td>9.50</td>
<td>9.50</td>
<td>5.6</td>
<td>5.6</td>
<td>5.6</td>
</tr>
<tr>
<td>576</td>
<td>9.22</td>
<td>9.67</td>
<td>9.72</td>
<td>9.67</td>
<td>4.8</td>
<td>5.4</td>
<td>4.8</td>
</tr>
<tr>
<td>640</td>
<td>9.40</td>
<td>9.80</td>
<td>9.80</td>
<td>9.80</td>
<td>4.3</td>
<td>5.3</td>
<td>4.3</td>
</tr>
<tr>
<td>704</td>
<td>9.55</td>
<td>9.91</td>
<td>10.05</td>
<td>9.91</td>
<td>3.8</td>
<td>5.2</td>
<td>3.8</td>
</tr>
<tr>
<td>768</td>
<td>9.67</td>
<td>10.00</td>
<td>10.17</td>
<td>10.00</td>
<td>3.5</td>
<td>5.2</td>
<td>3.5</td>
</tr>
<tr>
<td>832</td>
<td>9.77</td>
<td>10.15</td>
<td>10.21</td>
<td>10.15</td>
<td>3.9</td>
<td>5.1</td>
<td>3.9</td>
</tr>
<tr>
<td>896</td>
<td>9.86</td>
<td>10.29</td>
<td>10.36</td>
<td>10.29</td>
<td>4.4</td>
<td>5.1</td>
<td>4.4</td>
</tr>
<tr>
<td>960</td>
<td>9.95</td>
<td>10.40</td>
<td>10.43</td>
<td>10.40</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>1024</td>
<td>10.00</td>
<td>10.50</td>
<td>10.50</td>
<td>10.50</td>
<td>5.0</td>
<td>5.0</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Table 3: Comparisons with arbitrary probabilities

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

In Tables 2 and 3 we compare $c(T_H)$, $c(T_E)$, $c(T_{U_1})$ and $c(T_{U_2})$, for $n$ in the range 64 to 1024. In Table 2 we use uniform probabilities, and in Table 3, arbitrary probabilities, obtained from the standard Pascal generation routine for random numbers in the range 1 to 10000 (we found no significant variations by changing this range). All the probabilities were further normalized so that the total sum is 1. In Table 4 we compare the two heuristics with Huffman’s algorithm for $n$ in the range 1000 to 10000.

The main result observed in Table 2 is that, for uniform probabilities, Heuristics 2 equals the Exact Algorithm, while Heuristics 1 does not. The main explanation for this fact is that, when the Huffman tree is a complete binary tree, the improvements of Heuristics 2 apply very well. It can also be observed the small difference between Huffman’s algorithm and the other methods, and the decrease of the relative costs when $n$ increases. It can still be confirmed a theoretical result stated in [12]: the cost difference between the optimal even tree and the Huffman tree lays in the interval $[1/3, 1/2]$, being maximum $(1/2)$ when the number of symbols is $n = 2^k$ for some integer $k$, and minimum $(1/3)$ when $n = 3^k$.

Now, examine the results presented in Table 3, for arbitrary probabilities. First, compare data from Tables 2 and 3. We can see that all data in columns 2 to 5 in Table 3 are smaller than the corresponding ones in Table 2; this is an expected behavior, since the cost has the tendency to decrease as long as probabilities get unbalanced. The relative difference between $c(T_E)$ and $c(T_H)$ decreases considerably as $n$ increases. The same occurred for Heuristics 2, suggesting that it is also well applied for this situation, although it does not equal the optimal
Table 4: Comparisons with arbitrary probabilities

<table>
<thead>
<tr>
<th>$n$</th>
<th>$c(T_H)$ (A)</th>
<th>$c(T_{U_1})$ (B)</th>
<th>$c(T_{U_2})$ (C)</th>
<th>$\frac{\Delta c}{c}$ %</th>
<th>$\frac{C - c}{c}$ %</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>9.71</td>
<td>10.22</td>
<td>9.87</td>
<td>5.2</td>
<td>1.6</td>
</tr>
<tr>
<td>20000</td>
<td>10.72</td>
<td>11.21</td>
<td>10.87</td>
<td>4.6</td>
<td>1.5</td>
</tr>
<tr>
<td>30000</td>
<td>11.34</td>
<td>11.81</td>
<td>11.57</td>
<td>4.5</td>
<td>2.3</td>
</tr>
<tr>
<td>40000</td>
<td>11.72</td>
<td>12.22</td>
<td>11.87</td>
<td>4.3</td>
<td>1.3</td>
</tr>
<tr>
<td>50000</td>
<td>12.08</td>
<td>12.55</td>
<td>12.26</td>
<td>4.1</td>
<td>1.8</td>
</tr>
<tr>
<td>60000</td>
<td>13.04</td>
<td>13.55</td>
<td>13.25</td>
<td>3.9</td>
<td>1.6</td>
</tr>
<tr>
<td>70000</td>
<td>13.61</td>
<td>14.11</td>
<td>13.80</td>
<td>3.7</td>
<td>1.4</td>
</tr>
<tr>
<td>80000</td>
<td>14.65</td>
<td>14.55</td>
<td>14.25</td>
<td>3.6</td>
<td>1.5</td>
</tr>
<tr>
<td>90000</td>
<td>14.36</td>
<td>14.86</td>
<td>14.61</td>
<td>3.5</td>
<td>1.8</td>
</tr>
<tr>
<td>100000</td>
<td>14.62</td>
<td>15.12</td>
<td>14.81</td>
<td>3.4</td>
<td>1.3</td>
</tr>
<tr>
<td>110000</td>
<td>14.85</td>
<td>15.35</td>
<td>15.01</td>
<td>3.4</td>
<td>1.1</td>
</tr>
<tr>
<td>120000</td>
<td>15.05</td>
<td>15.55</td>
<td>15.25</td>
<td>3.3</td>
<td>1.4</td>
</tr>
<tr>
<td>130000</td>
<td>15.21</td>
<td>15.71</td>
<td>15.45</td>
<td>3.3</td>
<td>1.6</td>
</tr>
<tr>
<td>140000</td>
<td>15.36</td>
<td>15.86</td>
<td>15.61</td>
<td>3.3</td>
<td>1.7</td>
</tr>
<tr>
<td>150000</td>
<td>15.49</td>
<td>15.99</td>
<td>15.71</td>
<td>3.2</td>
<td>1.4</td>
</tr>
<tr>
<td>160000</td>
<td>15.62</td>
<td>16.12</td>
<td>15.80</td>
<td>3.2</td>
<td>1.2</td>
</tr>
<tr>
<td>170000</td>
<td>15.74</td>
<td>16.24</td>
<td>15.89</td>
<td>3.2</td>
<td>1.0</td>
</tr>
<tr>
<td>180000</td>
<td>15.85</td>
<td>16.35</td>
<td>16.01</td>
<td>3.2</td>
<td>1.0</td>
</tr>
<tr>
<td>190000</td>
<td>15.95</td>
<td>16.45</td>
<td>16.14</td>
<td>3.1</td>
<td>1.2</td>
</tr>
<tr>
<td>200000</td>
<td>16.04</td>
<td>16.54</td>
<td>16.25</td>
<td>3.1</td>
<td>1.3</td>
</tr>
<tr>
<td>210000</td>
<td>16.13</td>
<td>16.63</td>
<td>16.35</td>
<td>3.1</td>
<td>1.4</td>
</tr>
<tr>
<td>220000</td>
<td>16.21</td>
<td>16.74</td>
<td>16.45</td>
<td>3.1</td>
<td>1.5</td>
</tr>
<tr>
<td>230000</td>
<td>16.29</td>
<td>16.79</td>
<td>16.54</td>
<td>3.1</td>
<td>1.5</td>
</tr>
<tr>
<td>240000</td>
<td>16.36</td>
<td>16.86</td>
<td>16.61</td>
<td>3.1</td>
<td>1.6</td>
</tr>
</tbody>
</table>

solution. However, for Heuristics 1, the behavior is quite different. Both the absolute value of the difference to $c(T_H)$ and the relative value increased. Thus, Heuristics 2 outperforms Heuristics 1 in this situation.

Table 4 illustrates the costs obtained for large values of $n$ and arbitrary probabilities. The costs compared are $c(T_H)$, $c(T_{U_1})$ and $c(T_{U_2})$. The main results obtained from Table 3 are confirmed, that is, Heuristics 2 is far better than Heuristics 1. Moreover, the relative differences of costs from the two heuristics to Huffman’s algorithm again decrease. Those differences become negligible for large values of $n$.

Finally, from the three tables, we can observe a gap between the upper theoretical bound presented in Section 4 and the experimental results, since all the relative differences between the costs of the even trees obtained by Heuristics 2 and the Huffman trees were at most 5%, for $n$ large enough. It seems to be interesting to search for tighter bounds for this situation.
6.2 Experimental Results on Error Detection

Here we consider experimental results on error detection. Table 5 illustrates theoretical and experimental values for $P_m(T, b)$, which is the probability of error detection of an even tree, considering the model of section 5.

In the first column we present a range of values for $b$, the number of symbols in a message, varying from 10 to 5000. In the second column, the theoretical values for $P_m(T, b)$ are shown, according to the model presented in Section 5, using $Pt_e(T) = 0.1248$, which is nearly the average value for this parameter. The third column contains experimental values for $P_m(T, b)$, the probability of error detection, obtained via a simulation with an even tree $T$ having $Pt_e(T) = 0.1248$. The simulation involved the generation of about 20,000,000 random errors consisting of one to twenty changed bits.

<table>
<thead>
<tr>
<th>$b$</th>
<th>$P_m(T, b)$ Theoretical $(1 - (1 - 0.875^2)^b/(0.4992 + b))$</th>
<th>Experimental</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.852900</td>
<td>0.903540</td>
</tr>
<tr>
<td>25</td>
<td>0.922732</td>
<td>0.984620</td>
</tr>
<tr>
<td>50</td>
<td>0.959987</td>
<td>0.989553</td>
</tr>
<tr>
<td>100</td>
<td>0.979068</td>
<td>0.994164</td>
</tr>
<tr>
<td>250</td>
<td>0.991987</td>
<td>0.998482</td>
</tr>
<tr>
<td>500</td>
<td>0.995994</td>
<td>0.999150</td>
</tr>
<tr>
<td>1000</td>
<td>0.997907</td>
<td>0.999595</td>
</tr>
<tr>
<td>2500</td>
<td>0.999199</td>
<td>0.999831</td>
</tr>
<tr>
<td>5000</td>
<td>0.999599</td>
<td>0.999922</td>
</tr>
</tbody>
</table>

We can observe the fast growth of the error detection probability, as the length of the message increases. The error detection probability is fairly large for $b \geq 100$. It can also be verified that experimental values of error detection probabilities are greater than the predicted ones by the model, specially for lower values of $b$. This indicates that the error detection probability model can be enhanced. But the differences between theory and practice are quite low for $b > 100$.

7 Conclusions

Hamming posed the problem of integrating the advantage of the Huffman compression, with noise protection [11]. With this motivation, in this work we study even trees, where their symbols are assigned probabilities having arbitrary distributions. We propose an exact algorithm for constructing an optimal cost even tree, running in $O(n^3)$ time. Further, we describe two heuristics which require $O(n \log n)$ time for constructing a near optimal cost tree. We show that the cost
of a near optimal tree obtained by the second heuristics is at most 16.7\% higher than the cost of a Huffman’s tree, for the same set of symbols. Clearly, in terms of costs, an optimal tree lies between a Huffman’s tree and a near optimal tree.

The even trees constructed by the algorithms have the purpose of detecting the occurrence of errors in the binary message. Besides the study of the costs, we also describe a probabilistic model for estimating the ability of the trees to detect such errors. The bounds obtained suggest that the probability of the trees to produce $k$ consecutive wrong symbols is fairly small, for large messages. Finally, we report experimental results, with the aim of practically evaluating both the cost and error detection capability.

In fact, cost and error detection are opposite parameters. Within certain limits, one of these parameters can be improved, by making the other one worse. It would be interesting to extend this work, by examining codes with a better error detection capability, even sacrificing cost.

Acknowledgment

The authors would like to thank Artur Alves Pessoa for the insightful suggestions, which improved the exact dynamic programming algorithm of Section 2.

References


