A Nearly-optimal Fano-based Coding Algorithm

Luis G. Rueda*and B. John Oommen†

Abstract

Statistical coding techniques have been used for a long time in lossless data compression, using methods such as Huffman's algorithm, arithmetic coding, Shannon's method, Fano's method, etc. Most of these methods can be implemented either statically or adaptively. In this paper, we show that although Fano coding is suboptimal, it is possible to generate static Fano-based encoding schemes which are arbitrarily close to the optimal, i.e. those generated by Huffman's algorithm. By taking advantage of the properties of the encoding schemes generated by this method, and the concept of "code word arrangement", we present an enhanced version of the static Fano's method, namely Fano$^+$. We formally analyze Fano$^+$ by presenting some properties of the Fano tree, and the theory of list rearrangements. Our enhanced algorithm achieves compression ratios arbitrarily close to those of Huffman's algorithm on files of the Calgary corpus and the Canterbury corpus.

1 Introduction

1.1 Problem Statement

Huffman's algorithm is a well-known encoding method that generates an *optimal prefix* encoding scheme, in the sense that the average code word length is minimum. As opposed to this, Fano's method has not been used so much because it generates prefix encoding schemes that can be sub-optimal.

In this paper, we present an enhancement of the traditional Fano's method by which encoding schemes, which are arbitrarily close to the optimum, can be easily constructed.

1.2 Overview

We assume that we are given a source alphabet, $\mathcal{S} = \{s_i, \ldots, s_m\}$, whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, and a code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$. We intend to generate an encoding scheme, $\{s_i \rightarrow w_i\}$, in such a way that $\bar{\ell} = \sum_{i=1}^{m} p_i \ell_i$ is minimized, where ℓ_i is the length of w_i. Although, we specifically consider the binary code alphabet, the generalization to the r-ary case is not too intricate.

Lossless encoding methods used to solve this problem include Huffman's algorithm (Huffman, 1952), Shannon's method (Shannon and Weaver, 1949), arithmetic coding (Sayood, 2000), Fano's method (Hankerson

*Member, IEEE. School of Computer Science, University of Windsor, 401 Sunset Ave., Windsor, ON, N9B 3P4, Canada. E-mail: lrueda@uwindsor.ca. Partially supported by Departamento de Informática, Universidad Nacional de San Juan, Argentina, and NSERC, the Natural Science and Engineering Research Council of Canada. A preliminary version of this paper was presented at the 2001 IEEE Conference on Systems, Man and Cybernetics, Tucson, Arizon, USA.

†Fellow, IEEE. School of Computer Science, Carleton University, 1125 Colonel By Dr., Ottawa, ON, K1S 5B6, Canada. E-mail: oommen@scs.carleton.ca. Partially supported by NSERC, the Natural Science and Engineering Research Council of Canada.

et al., 1998), etc. Adaptive versions of these methods have been proposed, and can be found in (Faller, 1973; Gallager, 1978; Hankerson et al., 1998; Knuth, 1985; Rueda, 2002; Sayood, 2000). Our survey is necessarily brief as this is a well-reputed field.

We assume that the source is memoryless or zeroth-order, which means that the occurrence of the next symbol is independent of any other symbol that has occurred previously. Higher-order models include *Markov models* (Hankerson et al., 1998), *dictionary techniques* (Ziv and Lempel, 1977; Ziv and Lempel, 1978), *prediction with partial matching* (Witten et al., 1999), *grammar based compression* (Kieffer and Yang, 2000), etc., and the techniques introduced here are also readily applicable for such "structure" models.

1.3 Our Contribution

Huffman's algorithm proceeds by generating the so called Huffman tree, by recursively merging symbols (nodes) into a new *conceptual* symbol which constitutes an internal node of the tree. In this way, Huffman's algorithm generates the tree in a bottom-up fashion. As opposed to this, Fano's method proceeds by generating a coding tree as well, but it proceeds in a top-down fashion. At each step, the list of symbols is partitioned into two (or more, if the output alphabet is non-binary) new sublists, generating two or more new nodes in the corresponding coding tree. Although Fano's method, typically, generates a sub-optimal encoding scheme[1], the loss in compression ratio with respect to Huffman's algorithm can be relatively small, but can also be quite significant if the optimality of the partitioning is small.

The binary alphabet version of Fano's method proceeds by partitioning the list of symbols into two new sublists in such a way that the sums of probabilities of these two new sublists are as close to being equal as possible. This procedure is recursively applied to the new sublists until two atomic sublists with a single symbol are obtained, and simultaneously a bit is appended to the code words of each symbol in these sublists. As a result of this, if we start with the probabilities of occurrence in a decreasing order, it can be seen that, occasionally, symbols with higher probabilities are assigned to longer code words than those with lower probabilities. This condition is not desirable; we attempt to rectify this condition in Fano$^+$, a superior Fano-based scheme which we develop in this paper.

On the other hand, after constructing a coding tree (such as a Huffman tree or a Fano tree), an encoding scheme is generated from that tree by labeling the branches with the code alphabet (typically, binary) symbols. Given a tree constructed from a source alphabet of m symbols, 2^{m-1} different encoding schemes can be generated. Of these, only one of them is of a family of codes known as *canonical codes*, in which the code words are arranged in a lexicographical order (Witten et al., 1999). Canonical codes are desired because they allow *extremely* fast decoding, and require approximately *half* of the space used by a decoding tree. These codes are generated by Fano coding in a natural way.

In this paper, we introduce Fano$^+$, an enhanced version of the static Fano coding approach utilizing the concept which we called "code word arrangement", and which is based on a fundamental property of two lists arranged in an increasing and decreasing order respectively (Hardy et al., 1959). In our context, these lists are the code words (in terms of their lengths) and their probabilities respectively. This paper formally details the encoding scheme generation algorithm, the partitioning procedures suitable for Fano$^+$, and a rigorous analysis

[1] Even if optimal partition is in $\mathbf{P}$, Fano coding is still sub-optimal (Storer, 1988).

of their respective properties.

We finally discuss some empirical results obtained from running the static Huffman's algorithm, the static version of the traditional Fano coding, and Fano$^+$, on real life data. Our empirical results show that the compression ratios achieved by Fano$^+$ are comparable to those of other optimal encoding methods such as Huffman's algorithm. Although we use the zeroth order statistical model, other structure/statistical models such as higher-order models, dictionary models, etc., can also be used in conjunction with Fano$^+$ to achieve compression ratios that are *close to those attained by most well known compression schemes*.

2 Properties of the Traditional Fano Coding

Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ with probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq p_2 \geq \ldots \geq p_m$. Unless otherwise stated, in this paper, we assume that the code alphabet is $\mathcal{A} = \{0, 1\}$.

We define an encoding scheme as a mapping, $\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$, where $w_i \in \mathcal{A}^+$, for $i = 1, \ldots, m$. One of the properties of the encoding schemes generated by Huffman's algorithm is that $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$, where ℓ_i is the length of w_i. In general, this property is not satisfied by the encoding schemes generated by Fano's method. We also introduce a definition that is important to our work.

Definition 1. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, and let $\mathcal{A} = \{0, 1\}$ be the code alphabet. A *binary coding tree*, $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$, is a binary tree in which:

- (i) Every node has zero or two children.
- (ii) Every internal node, t_k, has a weight, τ_k, calculated as the sum of the weights of its two children.
- (iii) Every leaf, t_j, has a symbol, s_i, associated with it, and a weight, τ_j, which represents the probability of occurrence of s_i, namely p_i.

It is also important to mention that if the code word lengths are not in this order, the binary coding tree does not satisfy a fundamental property called the *sibling property* (Gallager, 1978; Hankerson et al., 1998), which is defined as follows.

Definition 2. Let $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ be a binary coding tree, whose weights are $\{\tau_1, \ldots, \tau_{2m-1}\}$, and where t_1 is the root node. $\mathcal{T}$ satisfies the *sibling property* if the following conditions hold:

- (i) $\tau_2 \geq \ldots \geq \tau_{2m-1}$.
- (ii) t_{2k} and t_{2k+1} are siblings in $\mathcal{T}$, for $k = 1, \ldots, m-1$.

Before we proceed with the theoretical analysis of the relation between the code word lengths and the sibling property, we recursively define the length of the path in a binary coding tree.

Definition 3. Consider a binary coding tree $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$ in which each leaf is associated with a source alphabet symbol, s_i. The code word length of a node t_j, where t_j is any node of the tree, is defined as follows:

- (i) 1 if t_j is a leaf, or

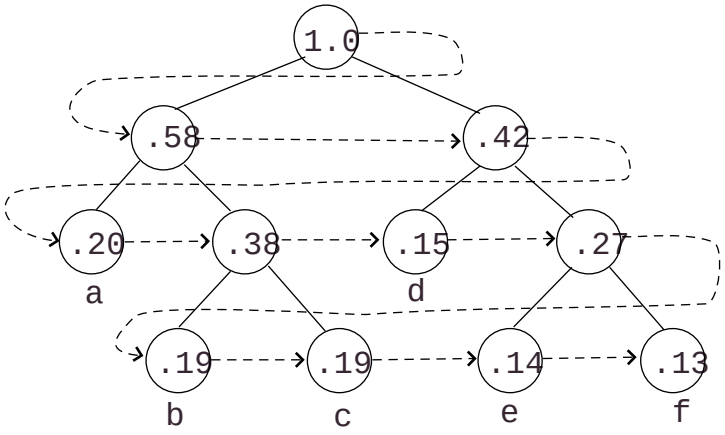


Figure 1: An example of a binary coding tree, which was constructed by following the principles of Fano coding, and does not satisfy the sibling property.

(*ii*) $1 + \text{length}(t_{j_c})$, where t_{j_c} is the left or right child of t_j, whenever t_j is an internal node.

Using this definition, we shall prove that whenever the probabilities of occurrences of the input symbols are in a non-increasing order, if the code words are in a non-decreasing order of length, the binary coding tree associated with such an encoding scheme does not satisfy the sibling property. This fact is clarified in Example 1 below and formally stated and proved in Theorem 1. It is thereafter used in Fano$^+$ to modify the binary coding tree so as to get an enhanced performance.

Example 1. Consider the source alphabet, $\mathcal{S} = \{a, b, c, d, e, f\}$, and the probabilities of occurrence $\mathcal{P} = [.2, .19, .19, .15, .14, .13]$. Suppose that a particular binary coding tree, $\mathcal{T}$, constructed from $\mathcal{S}$ and $\mathcal{P}$ following the principles of Fano coding is the one depicted in Figure 1.

Note that the list positions that we referred to as t_i, and the positions of the symbols of $\mathcal{S}$ in the list are t_{k_i}, which are listed in the following tables:

List Id	τ_i
t_1	1.00
t_2	.58
t_3	.42
t_4	.20
t_5	.38
t_6	.15
t_7	.27
t_8	.19
t_9	.19
t_{10}	.14
t_{11}	.13

t_{k_i}	k_i
t_{k_1}	4
t_{k_2}	8
t_{k_3}	9
t_{k_4}	6
t_{k_5}	10
t_{k_6}	11

The complete table of the indices, the probabilities of occurrence, $\mathcal{P}$, and the code word lengths, $\{\ell_i\}$, for the symbols are given in the following table:

s_i	p_i	ℓ_i
a	.2	2
b	.19	3
c	.19	3
d	.15	2
e	.14	3
f	.13	3

Observe that $\ell_i = 3 > 2 = \ell_j$, where $i = 3 < 4 = j$. The corresponding locations in the list are t_9 and t_6 whose weights are 0.19 and 0.15 respectively. The sibling property is violated, since $0.19 = p_3 > p_4 = 0.15$. □

Example 1 depicts a rather simple case, which is likely to occur when constructing the tree for real-life scenarios using Fano's method. We will observe this later when we present our empirical simulations on real-life files from standard benchmarks.

Theorem 1. Let $\mathcal{T}$ be a binary coding tree constructed from the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence $\mathcal{P} = [p_1, \ldots, p_m]$ are sorted, i.e. $p_1 \geq \ldots \geq p_m$. For any encoding scheme, $\phi : \mathcal{S} \rightarrow \{w_1, \ldots, w_m\}$, obtained after labeling as per the tree $\mathcal{T}$, where ℓ_i is the length of w_i, if there exist i and j such that $i < j$ and $\ell_i > \ell_j$, then $\mathcal{T}$ does not satisfy the sibling property.

Sketch of Proof. Let $\mathcal{T}^\star = \{t_1^\star, \ldots, t_{2m-1}^\star\}$ be a binary coding tree. $\mathcal{T}^\star$ satisfies the sibling property if and only if its nodes can be arranged in a sequence, $t_1^\star, t_2^\star, \ldots, t_{2m-1}^\star$, such that all non-root nodes of the form $t_{2i}^\star$ and $t_{2i+1}^\star$ are siblings in $\mathcal{T}^\star$, $i = 1, \ldots, m-1$, and $\tau_2^\star \geq \tau_3^\star \geq \ldots \geq \tau_{2m-1}^\star$.

Let $\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$ be an encoding scheme generated from the coding tree $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. Also, let ℓ_i is the length of w_i. Suppose that $\exists$ i and j, $i < j$, such that $\ell_i > \ell_j$. The proof is done by showing that the leaves associated with s_i and s_j, t_k and t_l, violate the sibling property for the three possible cases encountered:

(*i*) t_k and t_l are siblings.

(*ii*) t_k and t_l are not siblings, but they are in the same level.

(*iii*) t_k and t_l are not in the same level.

The details of the proof are omitted, and can be found in (Rueda, 2002; Rueda and Oommen, 2001). □

The reader will observe from Theorem 1 that when there is a leaf whose weight is greater than that of a leaf located in a lower level of the tree, the sibling property is not satisfied. As a consequence, the average code word length of any encoding scheme obtained from this tree is not minimal.

3 The Enhanced Coding Algorithm

Considering the facts discussed above, we now propose Fano$^+$ using the following modification to the traditional static Fano's method. It is well known that Fano's method requires that the source symbols and their

probabilities are sorted in a non-increasing order of the probabilities. What we incorporate is as follows: After all the code words are generated, we sort them in terms of their increasing order of lengths maintaining $\mathcal{S}$ and $\mathcal{P}$ in the order of the probabilities. This enhancement leads to Fano$^+$, a modified Fano coding algorithm which generates encoding schemes whose average code word lengths are arbitrarily close to those of the optimal ones (i.e. those generated by Huffman's algorithm). This enhancement is formalized in Rule 1 given below.

Rule 1. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq p_2 \geq \ldots \geq p_m$. Suppose that $\phi : s_1 \rightarrow w_1, \ldots, s_m \rightarrow w_m$ is the encoding scheme obtained by Fano's method. Rearrange $w_1, \ldots, w_m$ into $w'_1, \ldots, w'_m$ such that $\ell'_i \leq \ell'_j$ for all $i < j$, and simultaneously maintain $s_1, \ldots, s_m$ in the same order, to yield the encoding scheme: $\phi' : s_1 \rightarrow w'_1, \ldots, s_m \rightarrow w'_m$. □

The encoder construct a code, ϕ, using the static Fano's method, obtains the enhanced encoding scheme ϕ' by applying Rule 1. The decoder invokes the same rule obtaining ϕ', from which it generates the decoding scheme, $(\phi')^{-1} : w'_1 \rightarrow s_1, \ldots, w'_m \rightarrow s_m$.

In Example 1, swapping the symbols b and c (and updating the content of the tree), is equivalent to swapping their corresponding code words in any of the encoding schemes generated from $\mathcal{T}$. Observe that after this modification, $\mathcal{T}$ satisfies the sibling property. Note that we do not obtain the optimal encoding algorithm, like Huffman's, since we are only swapping code words, and hence the coding tree is the same as that constructed by Fano's method. There are some cases in which the structure of the tree must be changed so that it satisfies the sibling property.

The procedure for generating the encoding scheme (which also includes the sorting process) is given in Algorithm **Fano**$^+$ below.

The sorting procedure works, using the principles of *radix sorting* (Andersson et al., 1998), as follows. First, an array of $m-1$ elements is created, where the i^{th} cell allocates the code words of length ℓ_i. This is accomplished in the first **for** loop. In the second **for** loop, the entries of the array are scanned in canonical order, and for each entry (the inner **for** loop), the code words are "popped" from the array and allocated in the list of code words. As a result, the code words will be *sorted* in a non-decreasing order, and the probabilities of the symbols will *preserve* the original order, i.e. decreasing order.

Using the principles of radix sorting, an array of m integers in the range $1 \ldots k$, $k = O(m)$, can be sorted in $O(m)$ time and space. Note that in the sorting procedure, the sum of sizeOf(temp[i]) is actually the number of code words, which is m, and hence the worst-case time complexity of procedure sortSymbols(...) is $O(m)$. In the actual implementation, a pointer to each code word (as opposed to the code word itself) could be stored in array "term", in order to achieve *linear space* complexity.

Observe that Fano$^+$ yields a coding sequence for every source alphabet symbol. The actual encoding and decoding are both therefore straightforward. Elementary coding textbooks report how such table-based prefix codes can be implemented efficiently.

4 Properties of the Enhanced Fano Coding

To facilitate the analysis, we first introduce two important properties of Fano$^+$, the enhanced static Fano coding. The first relates to the efficiency in compression achieved by Fano$^+$, and the second is the property

Algorithm 1 Fano$^+$ // ***Enhanced_Fano_Coding***

Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
The code alphabet, $\mathcal{A} = \{0, 1\}$.
Output: The encoding scheme $\{s_i \rightarrow w_i\}$.
Method:
procedure Fano($\mathcal{S}, \mathcal{P}, \mathcal{A}$: list; **var** $\{w_i\}$: code)
Divide $\mathcal{S}$ and $\mathcal{P}$ into two sublists $\mathcal{S}_0, \mathcal{S}_1$ and $\mathcal{P}_0, \mathcal{P}_1$, such that the sum of probabilities of occurrence in each $\mathcal{P}_j$ is as nearly equal as possible.
for $j \leftarrow 0$ **to** 1 **do**
Push j to w_i for each $s_i \in \mathcal{S}_j$.
if $|\mathcal{S}_j| > 1$ **then**
Fano($\mathcal{S}_j, \mathcal{P}_j, \mathcal{A}$, $\{w_i\}$)
endif
endfor
procedure sortSymbols($\mathcal{S}, \{\ell_i\}$: list)
for $i \leftarrow 1$ **to** m **do**
Add w_i to temp[ℓ_i]
endfor
for $i \leftarrow 1$ **to** m **do**
for $j \leftarrow 1$ **to** sizeOf(temp[i]) **do**
$w_i \leftarrow$ temp[i][j]
endfor
endfor
endprocedure
Fano($\mathcal{S}$, $\mathcal{P}$, $\mathcal{A}$, $\{w_i\}$)
sortSymbols($\mathcal{S}$, $\{\ell_i\}$)
end Algorithm Fano$^+$ // ***Enhanced_Fano_Coding***

that it achieves lossless compression.

The efficiency of compression of Fano$^+$ is a direct consequence of the rearrangement of the code words such that they are sorted in an increasing order of length (Rule 1). This is stated in the following theorem, for which a proof can be found in (Hardy et al., 1959)[2].

Theorem 2. Let $p_1, \ldots, p_m$ be positive real numbers such that $p_1 \geq \ldots \geq p_m$, and let $\ell_1, \ldots, \ell_m$ be positive integers such that $\ell_1 \leq \ldots \leq \ell_m$. Then, for any rearrangement $\ell'_1, \ldots, \ell'_m$ of the list $\ell_1, \ldots, \ell_m$,

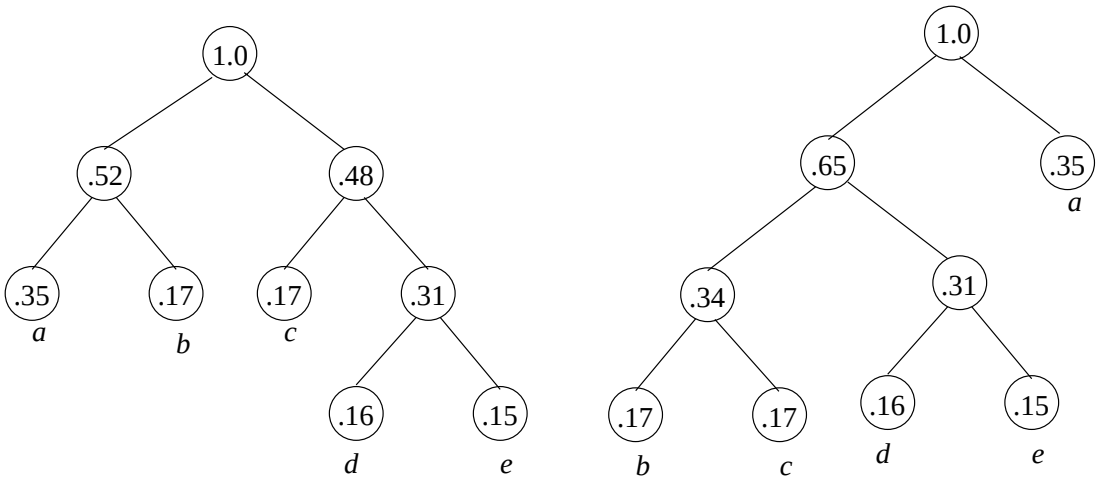
$$\sum_{i=1}^{m} p_i \ell_i \leq \sum_{i=1}^{m} p_i \ell'_i \,. \tag{1}$$

Observe that from Theorem 2, we can infer that for any list of code word lengths, we can obtain the optimal rearrangement, i.e. the one that satisfies $\ell_1 \leq \ldots \leq \ell_m$, by using Rule 1. However, this does not guarantee that we obtain the optimal encoding scheme (as in Huffman's algorithm), which also depends on constructing the optimal coding tree for a particular set of probabilities, $p_1, \ldots, p_m$.

The following counter-example shows that even the enhanced Fano coding algorithm does not ensure the optimal encoding scheme.

Example 2. Consider the source alphabet $\mathcal{S} = \{a, b, c, d, e\}$ whose probabilities of occurrence are $\mathcal{P} = [.35, .17, .17, .16, .15]$. The coding tree constructed using the conventional Fano's method is depicted in Figure 2 (a). The corresponding Huffman tree is depicted in Figure 2 (b). Observe that even after rearranging the code words obtained from labeling the tree of Figure 2 (a), it is not possible to achieve the optimal encoding

[2]The proof given in (Hardy et al., 1959) is also valid for all real, not necessarily positive $p_1, \ldots, p_m$ and $\ell_1, \ldots, \ell_m$.



(a) Using Fano's method. (b) Using Huffman's algorithm.

Figure 2: Two different binary coding trees constructed with the source alphabet and probabilities of occurrence given in Example 2.

scheme. This fact will also be observed in the empirical results shown later in this paper. □

To initiate discussions on the analysis of Fano$^+$, we first prove that it generates a prefix code. Note that this cannot be proven by using Kraft's inequality, since the latter is a *necessary* condition on the lengths of the code words, but *not sufficient* on the symbols composing the code words themselves.

Theorem 3. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be the source alphabet, and $\mathcal{A} = \{0, 1\}$ be the binary code alphabet. Procedure Fano(...) of Algorithm **Fano**$^+$ generates a prefix code.

Sketch of Proof. This theorem can be proven by an induction on the number of steps used in the partitioning. Let $\mathcal{S}(j)$ be the list being partitioned at time 'j', i.e. the j^{th} partitioning step of the invoked procedure Fano(...). At the time instant 'j', $\mathcal{S}(j)$ is partitioned into $\mathcal{S}_0(j)$ and $\mathcal{S}_1(j)$. The symbol '0' is added as a *suffix* to all the code words of $\mathcal{S}_0(j)$, and '1' is added as a *suffix* to all the codewords of $\mathcal{S}_1(j)$.

The basis case is straightforward, as it involves the initial partitioning in which $\mathcal{S}(1)$ is partitioned into $\mathcal{S}_0(1)$ and $\mathcal{S}_1(1)$, where the code words of $\mathcal{S}_0(1)$ start with '0' and the code words of $\mathcal{S}_1(1)$ start with '1'. Clearly, no code word of $\mathcal{S}_0(1)$ is a prefix of any code word in $\mathcal{S}_1(1)$ and vice versa.

The inductive step involves proving the hypothesis that no code word in $\mathcal{S}_0(j-1)$ is a prefix of *any* code word in $\mathcal{S}_1(j-1)$ and vice versa. This proof follows the exact same steps of the basis case in which we examine the symbols that occur after the $(j-1)^{st}$. The details are actually not too complicated, although the symbolic narration is cumbersome. The complete proof can be found in (Rueda, 2002; Rueda and Oommen, 2001). □

Theorem 4. Consider the source alphabet $\mathcal{S} = \{s_1, \ldots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \ldots, p_m]$, where $p_1 \geq \ldots \geq p_m$, and the binary code alphabet. Suppose that $\mathcal{X}$ is encoded into $\mathcal{Y}$ using Algorithm **Fano**$^+$. Then $\mathcal{X}$ is efficiently encoded into $\mathcal{Y}$, yielding an efficiency at least as that obtained using the conventional Fano's method.

Proof. The proof follows by a direct invocation of the properties of Theorems 1 and 2. □

File Name	Orig. Size: $l_{\mathcal{X}}$ (bytes)	Huffman		Fano		Fano$^+$	
		$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)
bib	111,261	73,003	34.38	73,133	34.26	73,010	34.37
book1	768,771	438,619	42.94	439,322	42.85	438,803	42.92
book2	610,856	368,587	39.66	369,691	39.47	369,537	39.50
geo	102,400	73,323	28.39	73,714	28.01	73,602	28.12
news	377,109	246,687	34.58	246,890	34.53	246,773	34.56
obj1	21,504	16,819	21.78	16,845	21.66	16,822	21.77
obj2	246,814	194,863	21.04	195,324	20.86	194,933	21.02
paper1	53,161	33,621	36.75	33,655	36.69	33,653	36.69
progc	39,611	26,189	33.88	26,355	33.46	26,326	33.53
progl	71,646	43,242	39.64	43,588	39.16	43,529	39.24
progp	49,379	30,480	38.27	30,527	38.17	30,501	38.23
trans	93,695	65,514	30.08	65,723	29.85	65,515	30.08
Total	2,547,207	1,610,947	36.76	1,614,767	36.61	1,613,004	36.68

Table 1: Empirical results obtained after compressing test files from the Calgary corpus by using the static Huffman's algorithm, the static Fano's method, and the sibling-enhanced version of the static Fano's method (Fano$^+$).

5 Empirical Results

In order to analyze the efficiency of Fano$^+$, we have conducted some experiments on files of the Calgary corpus[3] and the Canterbury corpus (Witten et al., 1999). The empirical results obtained are displayed in Tables 1 and 2 respectively. The columns labeled 'Huffman' and 'Fano' correspond to the static Huffman's algorithm and the traditional static Fano's method respectively. The columns labeled 'Fano$^+$' correspond to the sibling-enhanced version of the static Fano's method introduced in this paper. The columns labeled '$\ell_{\mathcal{Y}}$' represent the size (in bytes) of the compressed file. The columns labeled 'ρ' tabulate the percentage of compression obtained by the different methods, calculated as $\rho = \left(1 - \frac{\ell_{\mathcal{Y}}}{l_{\mathcal{X}}}\right) 100$, where $l_{\mathcal{X}}$ is the length of the input file. The last row contains the total for each column except for the column labeled 'ρ', for which the value of the cell corresponds to the *weighted average* of compression ratio.

Observe that the gain in percentage of compression is 0.06% and 0.07% on the files of the Calgary corpus and the Canterbury corpus respectively. For example, for the file *trans*, Huffman and Fano$^+$attain the same percentage of compression, e.g. 30.08%, whereas the traditional Fano coding compresses slightly less, e.g. 29.85%. Although the improved version of Fano's method does not guarantee the optimal encoding scheme, the weighted averages obtained from Fano$^+$ are significantly closer to those obtained by Huffman's algorithm. Observe also that in all the files, there is some gain in the compression ratio. This implies that in all the files encoded, $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_m$ was not satisfied by the encoding schemes generated by the traditional static Fano's method, and validates the results for such an enhancement.

We also observe that the compression ratios achieved by our implementation of Fano$^+$, *which are very close to the optimal*, can be significantly improved by incorporating higher-order structure models, such as dictionary based methods, Markovian models, etc. This is currently being investigated.

[3]Electronically available at `ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/`.

File Name	Orig. Size: $l_{\mathcal{X}}$ (bytes)	Huffman		Fano		Fano$^+$	
		$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)	$\ell_{\mathcal{Y}}$ (bytes)	ρ (%)
alice29.txt	148,481	84,765	42.91	85,254	42.58	84,999	42.75
asyoulik.txt	125,179	76,010	39.27	76,195	39.13	76,124	39.18
cp.html	24,603	16,456	33.11	16,477	33.02	16,458	33.10
fields.c	11,150	7,295	34.56	7,354	34.03	7,349	34.08
grammar.lsp	3,721	2,397	35.56	2,402	35.44	2,397	35.56
kennedy.xls	1,029,744	463,300	55.00	465,368	54.80	464,612	54.88
ice10.txt	419,235	244,124	41.76	244,197	41.75	244,141	41.76
plrabn12.txt	471,162	266,423	43.45	266,984	43.33	266,592	43.41
ptt5	513,216	107,027	79.14	107,138	79.12	107,075	79.13
sum	38,240	26,409	30.93	26,501	30.69	26,463	30.79
xargs.1	4,227	2,823	33.20	2,825	33.16	2,824	33.18
Total	2,788,958	1,297,029	53.49	1,300,695	53.36	1,299,034	53.42

Table 2: Details of the experiments performed with the static version of Huffman's algorithm, the traditional static Fano's method and the sibling-enhanced version of static Fano's method on files of the Canterbury corpus.

6 Conclusions

In this paper, we present an encoding scheme generation algorithm which is Fano-based and almost optimal. We first showed that for the encoding schemes, whose code words are not arranged in an increasing order of lengths, the corresponding coding tree does not satisfy the so-called sibling property. To rectify this, we introduced an enhanced version of the static Fano's method, Fano$^+$, whose properties have been formally proven.

The encoding algorithm associated with Fano$^+$ have been formally presented, rigorously analyzed, and empirically tested. Our empirical results on files of the Calgary corpus and the Canterbury corpus show that Fano$^+$ achieves percentages of compression which are almost optimal – very marginally below those of Huffman's algorithm.

The extension of Fano$^+$ for multi-symbol code alphabets follows directly from the binary solution proposed. The main problem in dealing with multi-symbol code alphabets is that a more elaborate partitioning procedure is required, which we propose in (Rueda, 2002). The zeroth-order model implemented here can also be extended to higher order models such as dictionary-based methods, Markovian models, etc., so as to achieve compression ratios comparable to those of the best state-of-art compression methods.

References

Andersson, A., Hagerup, H., Nilsson, S., and Raman, R. (1998). Sorting in Linear Time? *Journal of Computer and System Sciences*, 57:74–93.

Faller, N. (1973). An Adaptive System for Data Compression. *Seventh Asilomar Conference on Circuits, Systems, and Computers*, pages 593–597.

Gallager, R. (1978). Variations on a Theme by Huffman. *IEEE Transactions on Information Theory*, 24(6):668–674.

Hankerson, D., Harris, G., and Jr., P. J. (1998). *Introduction to Information Theory and Data Compression.* CRC Press.

Hardy, G., Littlewood, J., and Póyla, G. (1959). *Inequalities.* Cambridge University Press, 2nd. edition.

Huffman, D. (1952). A Method for the Construction of Minimum Redundancy Codes. *Proceedings of IRE*, 40(9):1098–1101.

Kieffer, J. C. and Yang, E. (2000). Grammar-Bassed Codes: A new Class of Universal Lossless Source Codes. *IEEE Transactions on Information Theory*, 46(3):737–754.

Knuth, D. (1985). Dynamic Huffman Coding. *Journal of Algorithms*, 6:163–180.

Rueda, L. (2002). *Advances in Data Compression and Pattern Recognition.* PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada.

Rueda, L. and Oommen, B. J. (2001). Nearly Optimal Fano-Based Canonical Codes. Technical Report SCS TR-01-05, School of Computer Science, Carleton University, Ottawa, Canada.

Sayood, K. (2000). *Introduction to Data Compression.* Morgan Kaufmann, 2nd. edition.

Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communications.* University of Illinois Press.

Storer, J. (1988). *Data Compression: Methods and Theory.* Computer Science Press.

Witten, I., Moffat, A., and Bell, T. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann, 2nd. edition.

Ziv, J. and Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343.

Ziv, J. and Lempel, A. (1978). Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory*, 25(5):530–536.