Nearly-optimal Fano-based Canonical Codes

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. Canonical codes, in which the code words are arranged in a lexicographical order, are advantageous because they can be decoded extremely expediently. Although Huffman's algorithm is optimal, the generation of a canonical Huffman code is not straightforward. Conversely, while the Fano coding is sub-optimal, it *can* lead to canonical codes.

In this paper, we resolve the dilemma by focusing on the static implementation of Fano's method. 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 Fano trees, and the theory of list rearrangements. Our enhanced algorithm achieves compression ratios arbitrarily close to those of Huffman's algorithm. Empirical results on files of the Calgary corpus and the Canterbury corpus corroborate the almost-optimal efficiency of our enhanced algorithm and its canonical nature. We believe that the compression efficiency of Fano$^+$ can be made to attain the compression ratios of the *best known schemes* if a structure model of the data is also incorporated.

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 are not optimal.

[*]School of Computer Science, Carleton University, 1125 Colonel By Dr., Ottawa, ON, K1S 5B6, Canada. E-mail: lrueda@scs.carleton.ca. Partially supported by Departamento de Informática, Universidad Nacional de San Juan, Argentina.

[†]Senior Member, 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.

On the other hand, canonical codes are a family of codes which are desired because they allow extremely fast decoding and simultaneously require less space than other codes. Generating a canonical code using Huffman coding involves a very complicated process – it first generates the code word lengths, and then uses these to derive the canonical code itself. As opposed to this, deriving a canonical code using Fano's method is straightforward – it is done in a fairly natural way by using the Fano tree itself.

In this paper, we present an enhancement of the traditional Fano's method in which canonical codes can be easily constructed, and, at the same time, the codes can be rendered to be arbitrarily close to the optimum, i.e. those generated by Huffman's algorithm.

1.2 Overview

The main problem in data compression involves taking an input sequence, $\mathcal{X} = x[1] \ldots x[M]$, where each $x[i]$ is drawn from a source alphabet, $\mathcal{S} = \{s_i, \ldots, s_m\}$, and transforming it into a, hopefully, smaller output sequence, $\mathcal{Y} = y[1] \ldots y[R]$, where each $y[i]$ is drawn from a code alphabet, $\mathcal{A} = \{a_1, \ldots, a_r\}$. Typically, the code alphabet is the binary alphabet which is the most widely used one in applications involving digital information processing.

When $\mathcal{X}$ is completely recoverable from $\mathcal{Y}$ by a process called *decompression*, we are talking about *lossless data compression*, which is the area in which we are interested. In particular, lossless compression methods include Huffman's algorithm [7], Shannon's method [13], arithmetic coding [12], Fano's method [14], etc. These are straightforward methods well known to even the novice in data compression, and so a survey of them is omitted. They can be found in [5, 12].

In *static coding*, the probabilities used to represent the distribution of the source are not modified in the entire encoding process. When these probabilities are changed (perhaps by a learning mechanism) during the encoding, the process is called *adaptive coding*. In this paper, we focus on the former. For the latter, many encoding techniques have been proposed in the literature, and can be found in [3, 4, 5, 9, 12].

Another classification that can be done in lossless compression is regarding the information used to encode a particular input sequence. The compression methods that we discuss use statistical information about the source. This information is referred to as the probability of occurrence of the source alphabet symbols, $\mathcal{P} = [p_1, \ldots, p_m]$. The methods that we discuss are *statistical*, since they use these pieces of statistical information.

Most of the encoding methods are based on a *replacement encoding scheme*. When we have an input sequence to be encoded, we build an encoding scheme, and the process of encoding is performed by replacing each source word symbol by a single code word based on the scheme. In this research work, we assume that the encoding process is done by starting from the first (the left most) symbol of $\mathcal{X}$, and continuing until the last (the right most) symbol. The decoding process is done in an analogous way.

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 occurred previously. Higher-order models include *Markov models* [5], *dictionary techniques* [16, 17], *prediction with partial matching* [2], *grammar based compression* [8], etc., and the techniques introduced here are also 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, in general, 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 [15]. Canonical codes are desired because they allow *extremely* fast decoding, and require approximately *half* of the space used by a decoding tree.

Obtaining canonical codes using Huffman's algorithm is not straightforward. In fact, it requires the execution of very complicated mechanisms, involving a three-phase process for calculating the code word lengths, and then invoking an extra process for obtaining the canonical code itself. Although Huffman's algorithm and

[1]This is due to the fact that the optimal partitioning in Fano's method is NP-hard [14].

Fano's method both take $O(m \log m)$ time[2], constructing the canonical code using the latter, is a straightforward matter. As mentioned above, Fano's method generates a coding tree in a top-down fashion. During the tree construction, the canonical code is obtained by assigning '0' to the left branches and '1' to right branches. Moreover, rearranging the code word symbols in an increasing order of length (the enhancement introduced here) can be done by rearranging the corresponding source alphabet symbols, again resulting in a canonical code.

In this paper, we introduce a front-end process to the static Fano encoding scheme, which achieves a code word rearrangement in such a way that the code word lengths are sorted in an increasing order of lengths. Indeed, we utilize the result of an archaic theorem concerning rearrangements of two sets [6] which, in our case, are the probabilities of the symbols and the lengths of the code words to yield a superior and enhanced performance.

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 (the code words in terms of their lengths and probabilities) arranged in an increasing and decreasing order respectively [6]. This paper details the encoding algorithm, the decoding algorithm, and the partitioning procedures suitable for Fano$^+$, and a rigorous analysis of their respective properties.

We finally discuss some empirical results obtained from running 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 of 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, $\phi : s_1 \to w_1, \ldots, s_m \to w_m$, 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.

[2]The most efficient mechanism for generating the canonical code using Huffman coding has been found to take $O(n \log \frac{m}{n})$ time and space, where n is the number of different probabilities [10], and hence, in the worst case, it takes $O(m \log m)$.

Definition 1. Let $\mathcal{S} = \{s_1, \dots , s_m\}$ be the source alphabet whose probabilities of occurrence are $\mathcal{P} = [p_1, \dots , p_m]$, and let $\mathcal{A} = \{0, 1\}$ be the code alphabet. A *coding tree*, $\mathcal{T} = \{t_1, \dots , t_{2m-1}\}$, 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 coding tree does not satisfy a fundamental property called the *sibling property.* 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 coding tree.

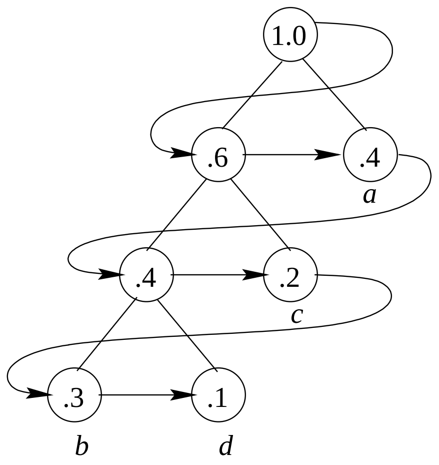
Definition 2. Consider a coding tree $\mathcal{T} = \{t_1, \dots , t_{2m-1}\}$ in which each leaf is associated with a source alphabet symbol, s_i. The code word length of a node t_j, where t_j is any node of the tree, is defined as follows:

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

(*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 not in an increasing order of length, the 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 coding tree so as to get an enhanced performance.

Example 1. Consider the source alphabet, $\mathcal{S} = \{a, b, c, d\}$, and the probabilities of occurrence $\mathcal{P} = [.4, .3, .2, .1]$. Suppose that a particular coding tree, $\mathcal{T}$, constructed from $\mathcal{S}$ and $\mathcal{P}$ is the one depicted in Figure 1.



List Id	τ_i
t_1	1.0
t_2	.6
t_3	.4
t_4	.4
t_5	.2
t_6	.3
t_7	.1

t_{k_i}	k_i
t_{k_1}	3
t_{k_2}	6
t_{k_3}	5
t_{k_4}	7

Figure 1: An example of a coding tree that does not satisfy the sibling property. Note that the list positions we referred to as t_i, and the position of the symbols of $\mathcal{S}$ in the list are t_{k_i}.

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	0.4	1
b	0.3	3
c	0.2	2
d	0.1	3

Observe that $\ell_i = 3 > 2 = \ell_j$, where $i = 2 < 3 = j$. The corresponding locations in the list are t_5 and t_6 whose weights are 0.2 and 0.5 respectively. Clearly, the sibling property is violated. Since $0.3 = p_2 > p_3 = 0.2$, Condition (ii) of the sibling property is not satisfied. $\square$

Example 1 depicts a rather simple case, which is not going to occur when constructing the tree for this scenario using Fano's method. However, when the number of symbols is large and the probabilities of occurrence are not skewed, there are some cases in which the coding tree constructed using Fano's method is similar to the one shown in Theorem 1.

Theorem 1. Let $\mathcal{T}$ be a 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, $\phi : \mathcal{S} \to \{w_1, \ldots, w_m\}$, 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.

Proof. Let $\mathcal{T}^\star = \{t_1^\star, \ldots, t_{2m-1}^\star\}$ be a 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 for any non-root node, the following conditions are satisfied:

(i) $t_{2i}^\star$ and $t_{2i+1}^\star$ are siblings in $\mathcal{T}^\star$, $i = 1, \ldots, m-1$, and

(ii) $\tau_2^\star \geq \tau_3^\star \geq \ldots \geq \tau_{2m-1}^\star$.

Let $\phi : s_1 \to w_1, \ldots, s_m \to w_m$, where ℓ_i is the length of w_i, be an encoding scheme generated from the coding tree $\mathcal{T} = \{t_1, \ldots, t_{2m-1}\}$. Suppose that $\exists\ i$ and j, $i < j$, such that $\ell_i > \ell_j$.

We know that every symbol of $\mathcal{S}$ is associated with a leaf of $\mathcal{T}$ and vice versa. Without loss of generality, suppose that s_i is associated with t_k, and s_j is associated with t_l. We consider three cases for the leaves, t_k and t_l.

(i) t_k and t_l are siblings. Using Definition 2, we see that ℓ_i should be equal to ℓ_j, and hence we need not consider this case, as it is excluded by virtue of the fact that we consider code words for which $\ell_i > \ell_j$ in this theorem.

(ii) t_k and t_l are not siblings, but they are in the same level. Again, from Definition 2, ℓ_i should be equal to ℓ_j. This implies that this case can also be excluded by virtue of the fact that we consider code words for which $\ell_i > \ell_j$ in this theorem.

(iii) t_k and t_l are not in the same level (obviously, they are not siblings). This leads us to four distinct cases for t_k and t_l enumerated below:

(a) $t_k = t_{2u}$ and $t_l = t_{2v}$ for some u and v, and both are the left children of a node, where ℓ_i is the depth of t_k and ℓ_j is the depth of t_k. Since $\ell_i > \ell_j$, the level in which t_k is located is greater than that of t_k. This implies that if we arrange the nodes from left to right so as to satisfy Condition (ii) of the sibling property, we obtain the following list:

$$t_1, \ldots, t_l, \ldots, t_k, \ldots, t_{2m-1} \ .$$

But since $p_i \geq p_j$, $\tau_k \geq \tau_l$, which violates Condition (ii) of the sibling property. The result follows.

The three remaining cases are proved in an analogous way. In order to avoid repetition, we only list them.

(b) $t_k = t_{2u}$ and $t_l = t_{2v+1}$, in which case t_k is a left child and t_l is a right child.

(c) $t_k = t_{2u+1}$ and $t_l = t_{2v}$, where t_k is a right child and t_l is a left child.

(d) $t_k = t_{2u+1}$ and $t_l = t_{2v} + 1$, which implies that both t_k and t_l are right children.

In each of this cases, as in Case (a), it can be seen that the node t_k precedes t_l in the list, and hence the sibling condition is violated.

The result follows. □

The proof of Theorem 1 illustrates the fact that when there is a leaf whose weight is greater than that of a leaf located in an upper 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 Algorithms

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

$\phi : s_1 \to w_1, \ldots, s_m \to w_m$

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 maintain $s_1, \ldots, s_m$ in the same order, obtaining the following encoding scheme :

$\phi' : s_1 \to w'_1, \ldots, s_m \to w'_m$. □

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

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. In the interest of completeness, the formal Algorithm **Enhanced_Fano_Encoding** and Algorithm **Enhanced_Fano_Decoding** are given below. Observe that unlike the encoding algorithm which does not explicitly create a Fano tree, to efficiently achieve decoding, the decoding algorithms also maintains this Fano tree. This is the version we give below, and leave the implementation details to the interested reader.

Algorithm 1 Enhanced_Fano_Encoding

```
Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
     The code alphabet, $\mathcal{A} = \{0, 1\}$. The source sequence, $\mathcal{X} = x[1] \dots x[M]$ .
Output: The output sequence, $\mathcal{Y} = y[1] \dots y[R]$ .
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 proba-
        bilities 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
  endfunction
   Fano($\mathcal{S}$, $\mathcal{P}$, $\mathcal{A}$, $\{w_i\}$)
   $\{w_1, \dots, w_m\} \leftarrow \{w'_1, \dots, w'_m\}$, where $\ell'_1 \leq \dots \leq \ell'_m$
   $k \leftarrow 1$
  for $i \leftarrow 1$ to $M$ do
     $y[k] \dots y[k + \ell_j - 1] \leftarrow w_j$, where $x[i] = s_j$
     $k \leftarrow k + \ell_j$
  endfor
end Algorithm Enhanced_Fano_Encoding
```

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 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,

Algorithm 2 Enhanced_Fano_Decoding

```
Input: The source alphabet, $\mathcal{S}$. The probabilities of occurrence, $\mathcal{P}$.
       The code alphabet, $\mathcal{A} = \{0, 1\}$. The encoded sequence, $\mathcal{Y} = y[1] \dots y[R]$ .
Output: The original source sequence, $\mathcal{X} = x[1] \dots x[M]$ .
Method:
  procedure Fano($\mathcal{S}, \mathcal{P}, \mathcal{A}$ : list; $n$ : node);
     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 proba-
         bilities of occurrence in each $\mathcal{P}_j$ is as nearly equal as possible.
     Create two new nodes: $n_0$ and $n_1$
     $L_n \leftarrow n_0$; $R_n \leftarrow n_1$
    for $j \leftarrow 0$ to 1 do
      $\varpi_{n_j} \leftarrow \sum_{p_i \in \mathcal{P}_j} p_i$
      if $|\mathcal{S}_j| > 1$ then
        Fano($\mathcal{S}_j, \mathcal{P}_j, \mathcal{A}, n_j$)
      else
        $\sigma_{n_j} \leftarrow \mathcal{S}_j$   // The symbol associated to the leaf $n_j$
      endif
    endfor
  endprocedure
   Create a new node: root
   $\varpi_{root} \leftarrow \sum_{i=1}^{m} p_i$
   Fano($\mathcal{S}$, $\mathcal{P}$, $\mathcal{A}$, root)
   $\forall i$ and $j$, such that $p_i > p_j$, swap $s_i$ and $s_j$ in the tree if $\ell_i > \ell_j$ .
   $k \leftarrow 1$; $n \leftarrow root$
  for $i \leftarrow 1$ to $R$ do
    if $y[i] = 0$ then
      $n \leftarrow L_n$
    else
      $n \leftarrow R_n$
    endif
    if $n$ =‘leaf’ then
      $x[k] \leftarrow \sigma_n$; $k \leftarrow k + 1$   // $\sigma_n$ is the symbol associated to node $n$
    endif
  endfor
end Algorithm Enhanced_Fano_Decoding
```

for which a proof can be found in [6][3].

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

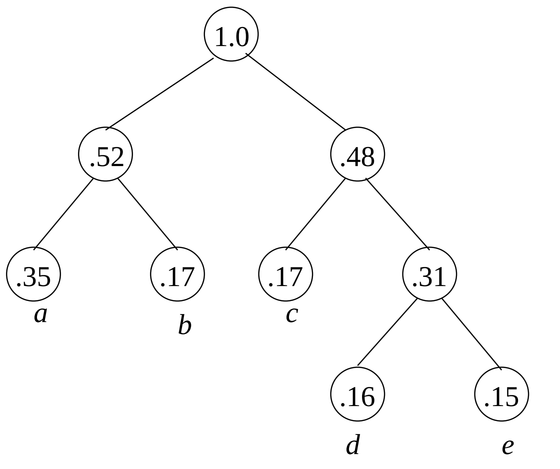
Prior to analyzing the correctness of Fano$^+$, and showing that it achieves lossless compression, we prove a result that will be needed later. This result is stated in Theorem 3, in which we prove that the enhanced static Fano's method generates a prefix code.

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 **Enhanced_Fano_Encoding** generates a prefix code.

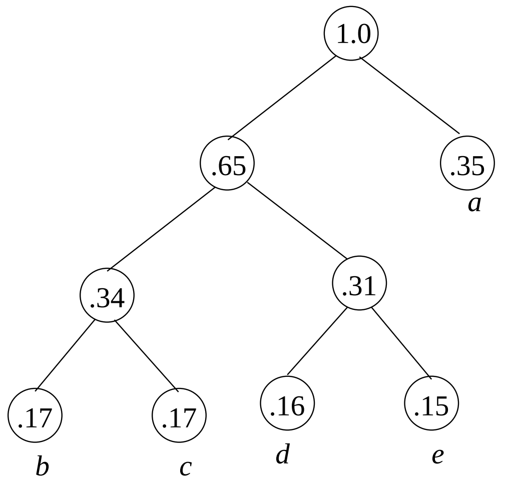
Proof. We prove this theorem by an induction on the number of steps used in the partitioning.

Let $\mathcal{S}(j)$ be the list being partitioned at time 'j', i.e. the j^{th} partitioning step of 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 $\mathcal{S}_1(j)$.

[3]The proof given in [6] 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 coding trees constructed with the source alphabet and probabilities of occurrence given in Example 2.

The basis case is straightforward, as it involves the first 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.

We now proceed to the inductive step in which the hypothesis is 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.

We have three mutually exclusive and exhaustive cases, which are the following:

(*i*) $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}\}$, where both $\mathcal{S}_0(j)$ and $\mathcal{S}_1(j)$ have the cardinality unity.

(*ii*) $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}, \ldots, s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$.

(*iii*) $\mathcal{S}_0(j) = \{s_i, \ldots, s_{i+v-1}\}$ and $\mathcal{S}_1(j) = \{s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$.

We consider these three cases separately.

(*i*) In this case, the two sublists resulting from the partitioning of $\mathcal{S}(j)$ are $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}\}$.

First of all, from the inductive hypothesis, no code word in $\mathcal{S}_0(j-1)$ is a prefix of any code word of $\mathcal{S}_1(j-1)$, and vice versa. We now have to deal with $\mathcal{S}(j)$ which is either $\mathcal{S}_0(j-1)$ or $\mathcal{S}_1(j-1)$. Since '0' is added as a *suffix* to w_i (the code word corresponding to s_i), and '1' is added as a *suffix* to w_{i+1} (the code word corresponding to s_{i+1}), it necessarily means that w_i has the form $w_{i_1} \dots w_{i_n}0$ and w_{i+1} has the form $w_{i_1} \dots w_{i_n}1$, where $w_{i_1}, \dots, w_{i_n} \in \mathcal{A}$. This ensures that neither w_i nor w_{i+1} are prefixes of each other. This completes this case.

(ii) In this case, the two sublists resulting from the partitioning of $\mathcal{S}(j)$ are of the form $\mathcal{S}_0(j) = \{s_i\}$ and $\mathcal{S}_1(j) = \{s_{i+1}, \dots, s_{i+v}\}$, where $v = |\mathcal{S}(j)| - 1$. As mentioned above, as a consequence of the inductive hypothesis, no code word of $\mathcal{S}(j)$ is a prefix of any code words that are not in $\mathcal{S}(j)$, and vice versa.

In addition, after '0' is added as a *suffix* to w_i (the code word corresponding to s_i) and '1' is added as a *suffix* to $w_{i+1}, \dots, w_{i+v}$ (the code words corresponding to $s_{i+1}, \dots, s_{i+v}$), w_i has the form $w_{i_1} \dots w_{i_n}0$, and all the code words in $\mathcal{S}(j)$, $w_{i+1}, \dots, w_{i+v}$, have the prefix $w_{i_1} \dots w_{i_n}1$, where $w_{i_1}, \dots, w_{i_n} \in \mathcal{A}$. This ensures that that neither w_i nor $w_{i+1}, \dots, w_{i+v}$ are prefixes of each other. The inductive step for this case follows.

The partitioning stops for $\mathcal{S}_0(j)$, since $|\mathcal{S}_0(j)| = 1$. For $\mathcal{S}_1(j)$, the partitioning is recursively invoked, leading again to either Case (i), (ii), or (iii).

(iii) As a result of the partitioning of $\mathcal{S}(j)$, $\mathcal{S}_0(j) = \{s_i, \dots, s_{i+v-1}\}$ and $\mathcal{S}_1(j) = \{s_{i+v}\}$ are obtained, where $v = |\mathcal{S}(j)| - 1$. The analysis of this case is identical (except that the suffix of '0' becomes the suffix of '1' and vice versa) to the analysis of Case (ii) above, and omitted to avoid repetition.

From the three cases analyzed above, the induction is achieved. Thus, the procedure Fano(...) of Algorithm **Enhanced_Fano_Coding** generates a prefix code for every partitioning step 'j', and the result follows. $\square$

Theorem 4. Consider the source alphabet $\mathcal{S} = \{s_1, \dots, s_m\}$ whose probabilities of occurrence are $\mathcal{P} = [p_1, \dots, p_m]$, where $p_1 \geq \dots \geq p_m$, and the binary code alphabet. Suppose that $\mathcal{X}$ is encoded into $\mathcal{Y}$ using Algorithm **Enhanced_Fano_Coding**, then

(i) $\mathcal{X}$ is efficiently encoded into $\mathcal{Y}$, yielding an efficiently at least as that obtained using the conventional Fano's method.

(ii) If $\mathcal{Y}$ is decoded into $\mathcal{X}^\star$ using Algorithm **Enhanced_Fano_Decoding**, then $\mathcal{X}^\star = \mathcal{X}$.

Proof. We proceed by dividing the encoding and decoding algorithms into two sequential phases. The first consists of the traditional Fano coding itself, and the second is the code word rearrangement as per Rule 1.

Let us consider the encoding algorithm. The first phase involves the creation of a code, $\phi : s_1 \rightarrow w_1, \dots, s_m \rightarrow w_m$, using the traditional Fano coding.

In the second phase of the encoding, the code words of ϕ are rearranged using Rule 1, obtaining $\phi' : s_1 \to w'_1, \dots, s_m \to w'_m$, where $w'_1, \dots, w'_m$ is a permutation or rearrangement of $w_1, \dots, w_m$, such that $\ell'_1 \leq \dots \leq \ell'_m$.

We now observe that $w'_1, \dots, w'_m$ is a sequence of code words that satisfies $\ell'_1 \leq \dots \leq \ell'_m$. Since $\ell_1, \dots, \ell_m$ is a rearrangement of the lengths $\ell'_1, \dots, \ell'_m$, using the result of Theorem 2, it is clear that $\bar{\ell}' = \sum_{i=1}^m p_i \ell'_i \leq \sum_{i=1}^m p_i \ell_i = \bar{\ell}$. This proves the first assertion that the encoding scheme obtained after the rearrangement, ϕ', is at least as efficient as the original encoding scheme, ϕ.

We now consider Assertion (ii).

Suppose now that $\mathcal{Y}$ is decoded into $\mathcal{X}^\star$ using Algorithm **Enhanced_Fano_Decoding**. In the first phase of the decoding process, a Fano tree is constructed using the conventional Fano coding, by procedure Fano(...), after which the actual decoding is achieved.

We, first of all, emphasize the duality of the encoding/decoding processes when a rearrangement is done. The effect of a rearrangement in the decoding process can be easily seen to obey the rule : Swapping two symbols in the tree, from $w_i \to s_i$ and $w_j \to s_j$, is equivalent to swapping the two symbols in the original scheme as $w_i \to s_j$ and $w_j \to s_i$. This is equivalent to swapping two code words in the *encoding* scheme generated in the encoding process : The swap of the pair $s_i \to w_i$ and $s_j \to w_j$ is equivalent to $s_i \to w_j$ and $s_j \to w_i$.

In the encoding, ϕ is generated by assigning (adding as a suffix) '0' to the code words corresponding to the symbols of the first sublist, $\mathcal{S}_0$, and '1' to the code words corresponding to the symbols of the second sublist, $\mathcal{S}_1$. This is equivalent to labeling the tree constructed in the decoder by assigning '0' to the left branches, and '1' to the right branches[4]. Using this tree and this predetermined labeling strategy to decode $\mathcal{Y}$, is equivalent to having a decoding scheme $\phi^{-1} : w_1 \to s_1, \dots, w_m \to s_m$.

From Theorem 3, we know that ϕ is a prefix code. This uniquely decodability property, in turn, implies that for $w_1, \dots, w_m$, no code word w_i is prefix of any other code word w_j, for all i and j, $i \neq j$. Clearly, any rearrangement of $w_1, \dots, w_m$ also satisfies the prefix property, and so this is also true for the enhanced coding : $\phi : s_1 \to w'_1, \dots, s_m \to w'_m$. Consequently, by proceeding from left to right, $\mathcal{X}$ is encoded into $\mathcal{Y}$ by using a uniquely decodable code, ϕ' [5]. This uniquely decodability property, in turn, implies that if $\phi'(\mathcal{X}) = \mathcal{Y}$, then $(\phi')^{-1}(\mathcal{Y}) = \mathcal{X}^\star = \mathcal{X}$, where $(\phi')^{-1} : w'_1 \to s_1, \dots, w'_m \to s_m$. This proves the second assertion.

The theorem is thus proved. □

Remark 1. It is important to observe that, as pointed out in the proof of Theorem 4, when swapping two source alphabet symbols instead of the two corresponding code words, the code generated in the first phase

[4]This is an arbitrary labeling strategy. Any of the 2^{m-1} different strategies can also be used, whenever the encoder and the decoder are agreed on using the same labeling strategy.

preserves the property of being a canonical code, even after the rearrangement. This feature of Fano$^+$ is of fundamental importance, since, as mentioned earlier, canonical codes are desired because of their superior decoding speed, and the substantial memory savings that they yield. Moreover, this fundamental property of Fano$^+$ is performed without adding any extra computational complexity in either time or space.

We now present the empirical results demonstrating the efficiency of Fano$^+$, and comparing it to the relative efficiency of the Fano and Huffman algorithms.

5 Empirical Results

One of the set of files used to evaluate the performance of the compression algorithms presented in this research work was obtained from the University of Calgary. It has been universally used for many years as a standard test suite and is known as the Calgary Corpus[5]. The other set of files was obtained from the University of Canterbury. This benchmark, known as Canterbury Corpus, was proposed in 1997 as a replacement for the Calgary corpus test suite [1]. These files are widely used to test and compare the efficiency of different compression methods. They represent a wide range of different applications such as executable programs, spreadsheets, web pages, pictures, source code, postscript source, etc.

In order to analyze the efficiency of the improved version of static Fano's method (Fano$^+$), we have conducted some experiments on files of the Calgary corpus and the Canterbury corpus. 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 sequence. 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. 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.

[5]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 (%)
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	49.92
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$^+$).

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.

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.

6 Conclusions

In this paper, we attempt to derive a canonical code 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 sibling property. To rectify this, we introduced an enhanced version of the static Fano's method, Fano$^+$.

One of the most important features of Fano$^+$ is that it generates a canonical code without increasing the computational complexity in either time or space. It is well known that generating a canonical code using the Huffman coding requires extremely complicated processes. This is rendered straightforward in Fano$^+$. Note that this is significant since canonical codes are of fundamental importance in decoding, since they allow extremely fast decoding, and require half as much space as that used by the decoding tree.

The encoding and decoding algorithms 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. Furthermore, it is computationally superior, and is thus a desirable option when compared to the latter.

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 [11]. 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

[1] R. Arnold and T. Bell. A Corpus for the Evaluation of Lossless Compression Algorithms. pages 201–210, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[2] J. Cleary and I. Witten. Data Compression Using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

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

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

[5] D. Hankerson, G. Harris, and P. Johnson Jr. *Introduction to Information Theory and Data Compression.* CRC Press, 1998.

[6] G. Hardy, J. Littlewood, and G. Póyla. *Inequalities.* Cambridge University Press, 2nd edition, 1959.

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

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

[9] D. Knuth. Dynamic Huffman Coding. *Journal of Algorithms*, 6:163–180, 1985.

[10] A. Moffat and A. Turpin. Efficient Construction of Minimum-redundancy Codes for Large Alphabets. *IEEE Transactions on Information Theory*, 44(4):1650–1657, July 1998.

[11] L. Rueda. *Enhanced Fano-based Coding and Optimal Pairwise Linear Classification.* PhD thesis, School of Computer Science, Carleton University, Ottawa, Canada. In preparation. (Thesis title is tentative).

[12] K. Sayood. *Introduction to Data Compression.* Morgan Kaufmann, 2nd. edition, 2000.

[13] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communications.* University of Illinois Press, 1949.

[14] J. Storer. *Data Compression: Methods and Theory.* Computer Science Press, 1988.

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

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

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