Lossless Data Compression via Substring Enumeration

Danny Dubé   Vincent Beaudoin
Université Laval, Canada
Email: `Danny.Dube@ift.ulaval.ca`   `Vincent.Beaudoin.1@ulaval.ca`

Abstract

We present a technique that compresses a string w by enumerating all the substrings of w. The substrings are enumerated from the shortest to the longest and in lexicographic order. Compression is obtained from the fact that the set of the substrings of a particular length gives a lot of information about the substrings that are one bit longer. A linear-time, linear-space algorithm is presented. Experimental results show that the compression efficiency comes close to that of the best PPM variants. Other compression techniques are compared to ours.

1 Basic Idea

We propose a technique of *lossless data compression via substring enumeration* (CSE) that compresses a string of bits $\mathbf{D}$ in three steps: first, it builds a tree that counts the number of occurrences of each of $\mathbf{D}$'s substrings, while considering $\mathbf{D}$ to be circular; second, it enumerates all of $\mathbf{D}$'s substrings, from the shortest to the longest and in lexicographic order, and, third, it indicates which of the full-length substrings is $\mathbf{D}$.

If the document to be compressed is over an alphabet other than $\{0, 1\}$, it can trivially be encoded symbol by symbol into bits using a fixed-length code. Throughout the paper, we assume that $\mathbf{D}$ is over $\{0, 1\}$. Consequently, when the document is too long, it can be divided into blocks and each block can then be compressed using substring enumeration. So we assume that $\mathbf{D}$, whether it is the complete document or a block, needs not be divided in parts. We further assume that $\mathbf{D}$ is non empty. Finally, for reasons that we give in Section 4, we assume that $\mathbf{D}$ is not made from k repetitions of some string, for any $k > 1$.

The enumeration proceeds by describing the set of substrings of length L, for L going from 1 to N, where N is $\mathbf{D}$'s length. Given a particular L, the description consists in indicating the number of occurrences of each possible L-bit substring of $\mathbf{D}$. This description does *not* enumerate the substrings in the order in which they appear in $\mathbf{D}$ but rather in lexicographic order. For example, let $\mathbf{D}$ be '`01000001`'. Figure 1 presents all the substrings of lengths 1 to 8. Note that, naturally, one of the 8-bit substrings happens to be $\mathbf{D}$ itself. This is why CSE's last step consists in transmitting the rank of the correct rotation.

We do not yet explain how the numbers of occurrences ought to be encoded. However, a key observation is that the enumeration of the L-bit substrings gives a lot of information about the enumeration of the $(L + 1)$-bit ones. It is this observation that leads to the ability to compress data. If one were to enumerate substrings without taking the shorter ones into account, one would emit $O(N^2)$ numbers of occurrences. Such a naïve enumeration would have no chance of effectively compressing $\mathbf{D}$.

Length	**Substrings**							
1	6×0						2×1	
2	4×00				2×01		2×10	
3	3×000			1×001	2×010		1×100	1×101
4	2×0000		1×0001	1×0010	1×0100	1×0101	1×1000	1×1010
5	1×00000	1×00001	1×00010	1×00101	1×01000	1×01010	1×10000	1×10100
6	1×000001	1×000010	1×000101	1×001010	1×010000	1×010100	1×100000	1×101000
7	1×0000010	1×0000101	1×0001010	1×0010100	1×0100000	1×0101000	1×1000001	1×1010000
8	1×00000101	1×00001010	1×00010100	1×00101000	1×01000001	1×01010000	1×10000010	1×10100000

Figure 1: Substring enumeration for '`01000001`'

2 Definitions

2.1 Notation

Throughout the paper, we follow the following conventions about the use of variables, unless stated otherwise: Σ is an alphabet; $\{0,1\}$ is the binary alphabet; a, b, and c are bits, i.e. $a, b, c \in \{0,1\}$; $\mathbf{D}$, u, v, w, and x are strings of bits; N is $\mathbf{D}$'s length; p is the position of an occurrence of a substring in $\mathbf{D}$; L is used to denote both the length of a string and the number of a level in a tree, where the root is considered to lie at level 0; C, with indices and possibly exponents, is an integer that counts the number of occurrences of some substring; T and t, possibly with indices, are trees; and m and n, with indices, are tree nodes.

We denote the length of w by $|w|$. We denote the empty string by ϵ. We denote the negation of b by $\bar{b}$. We also need the *infinite repetition*: c^∞ is an infinite string made up of the symbol $c \in \Sigma$ repeated infinitely often. Likewise, Σ^∞ is the set of all infinite strings over Σ and w^∞ is an infinite string over Σ which is the concatenation of an infinite number of copies of w, provided $w \neq \epsilon$.

2.2 Occurrences

We need to be careful with the definition of "occurrence" when we think of $\mathbf{D}$ as being circular. If, say, '`01`' appears somewhere in $\mathbf{D}$, one could argue that '`01`' would also appear N bits further, and $2N$ bits further, etc. If a substring were to appear at all in $\mathbf{D}$, then it would appear infinitely often. Such a definition would not be useful, in our case. Instead, we say that w *occurs* at *position* p if:

$$\exists\ u \in \{0,1\}^*,\ v \in \{0,1\}^\infty.\ |u| = p < N \ \text{ and } \ u\,w\,v = \mathbf{D}^\infty.$$

The *number* of occurrences of w in $\mathbf{D}$ is the number of positions where w occurs. Note that positions are required to be between 0 and $N-1$. This ensures that a single occurrence is not counted multiple times only because of $\mathbf{D}$'s circularity. Note also that we do *not* limit the length of $\mathbf{D}$'s substrings. In particular, a substring can be longer than $\mathbf{D}$. It can also be the empty string ϵ.

Note that, if $w\,b$ occurs at position p in $\mathbf{D}$, then so does w. If $b\,w$ occurs at p, then w occurs at $(p+1) \bmod N$. If w occurs at p, then there exist b and c such that $w\,b$ occurs at p and $c\,w$ occurs at $(p-1) \bmod N$.[1]

[1]Due to lack of space, we do not include demonstrations in the paper.

3 Butterflies

3.1 Trace of the enumeration for our example

Before we explain the strange title of this section, we start by tracing and commenting the process of transmitting the numbers of occurrences of **D**'s substrings. **D** is the one in our example, with numbers as showed in Figure 1. In the following trace, we denote the number of occurrences of w by C_w. Our comments on the transmission process are made according to the point of view of the decompressor.

We first consider the sole substring of length 0, that is, ϵ. Counter C_ϵ, which is $N = 8$, has to be handled specially. Depending on the context, the decompressor might know N in advance exactly, partially, or not at all. In one way or another, the decompressor must be made aware of C_ϵ.

Next, we consider the substrings of length 1. We focus on C_{0}. Note that, once C_{0} is known, the decompressor can deduce the value of C_{1}. Given what has been transmitted by now, the decompressor only knows that $0 \leq C_{\mathsf{0}} \leq 8$.[2] Consequently, $C_{\mathsf{0}} = 6$ is transmitted subject to $0 \leq C_{\mathsf{0}} \leq 8$. This implies that $C_{\mathsf{1}} = 2$.

Now, on level 2, we start with C_{00} and C_{01}. We know that $C_{\mathsf{00}} + C_{\mathsf{01}} = C_{\mathsf{0}} = 6$. By taking no other information into account, it would seem that C_{00} lies between 0 and 6, inclusively. However, we must notice that $2 = C_{\mathsf{1}} \geq C_{\mathsf{01}}$. The bound on C_{01} forces C_{00} to lie only between 4 and 6, inclusively. (The symmetric observation, that is, $8 = C_{\mathsf{0}} \geq C_{\mathsf{00}}$, does not constrain C_{00} any further.) So $C_{\mathsf{00}} = 4$ is transmitted subject to $4 \leq C_{\mathsf{00}} \leq 6$, allowing the decompressor to also determine that $C_{\mathsf{01}} = 2$.

Finally, we look at one more step of the trace by considering C_{10} and C_{11}. We know that $C_{\mathsf{10}} + C_{\mathsf{11}} = C_{\mathsf{1}} = 2$. This suggests that $0 \leq C_{\mathsf{10}} \leq 2$. Like in the preceding paragraph, we can take into account the facts that $6 = C_{\mathsf{0}} \geq C_{\mathsf{10}}$ and $2 = C_{\mathsf{1}} \geq C_{\mathsf{11}}$ but these do not constrain C_{10} further. Consequently, it seems that about $\log_2 3$ bits need be transmitted to describe C_{10}'s value. However, it is crucial to also take into account the fact that $C_{\mathsf{01}} + C_{\mathsf{11}} = C_{\mathsf{1}} = 2$. Thanks to this observation, the decompressor can determine that $C_{\mathsf{11}} = 0$, and then that $C_{\mathsf{10}} = 2$, *without requiring any bit to be transmitted.*

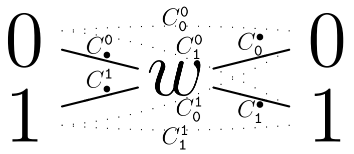
3.2 Bidirectional prediction

When thinking about the idea of prediction, it is natural to think about the equation $C_w = C_{w\mathsf{0}} + C_{w\mathsf{1}}$. However, in the last step of the above trace, we also used, quite profitably, the equation $C_w = C_{\mathsf{0}w} + C_{\mathsf{1}w}$. The combined use of these two equations leads to our notion of *butterfly.*

Let us consider a step where we are about to transmit the number of occurrences of a substring x, where $|x| \geq 2$. We can view x as a *core* w with one bit added at each end; i.e. $x = a\,w\,b$. It is then natural to refer to the substrings x is related to: $a\,w\,\bar{b}$, $\bar{a}\,w\,b$, and $\bar{a}\,w\,\bar{b}$. We say that a core w with a bit added at each end is an *extension* of w. Figure 2a, depicts the four extensions of w, each of which corresponding to a path from a bit on the left to a bit on the right. We can see (with a bit of imagination) the shape of a butterfly.[3]

[2]In fact, if C_{0} was one of 0 or 8, then **D** would be repetitive. However, we assumed earlier that **D** was not. Still, we do not try to use non-repetitiveness in order to narrow down the set of possible values for the counters.

[3]Recall that, at this point, C_w, $C_{\mathsf{0}w}$, $C_{\mathsf{1}w}$, $C_{w\mathsf{0}}$, and $C_{w\mathsf{1}}$ are known but not $C_{\mathsf{0}w\mathsf{0}}$, $C_{\mathsf{0}w\mathsf{1}}$, $C_{\mathsf{1}w\mathsf{0}}$, nor



Counter	$C_\bullet^\bullet$	$C_\bullet^0$	$C_\bullet^1$	$C_0^\bullet$	$C_1^\bullet$	C_0^0	C_1^0	C_0^1	C_1^1
# occur. of	w	$0w$	$1w$	$w0$	$w1$	$0w0$	$0w1$	$1w0$	$1w1$

Figure 2: a) A butterfly and the four extensions of w; b) counters for the substrings

In Figure 2a, each of the 8 links are labeled with counters. The meaning of these counters is summarized in Figure 2b. The counters are related as follows.

$$C_\bullet^\bullet \;=\; C_\bullet^0 + C_\bullet^1 \;=\; C_0^\bullet + C_1^\bullet$$
$$C_\bullet^0 \;=\; C_0^0 + C_1^0 \qquad C_\bullet^1 \;=\; C_0^1 + C_1^1 \qquad C_0^\bullet \;=\; C_0^0 + C_0^1 \qquad C_1^\bullet \;=\; C_1^0 + C_1^1$$

When enumerating the counters for the L-bit substrings of $\mathbf{D}$, it is sub-optimal to proceed by considering an $(L-1)$-bit "parent" substring w and transmitting the counters C_{w0} and C_{w1} of its "children". Instead, it is preferable to consider an $(L-2)$-bit core w and transmit the counters C_{0w0}, C_{0w1}, C_{1w0}, and C_{1w1} of its four extensions at once.

Among the counters used in a butterfly, $C_\bullet^\bullet$ is the core's one, $C_\bullet^0$, $C_\bullet^1$, $C_0^\bullet$, and $C_1^\bullet$ are the ones of w's "partial" extensions, and C_0^0, C_1^0, C_0^1, and C_1^1 are the ones of w's extensions. The latter are the only ones that are not known yet. Note that learning the value of one of the unknown counters determines the values of the three others. We choose C_0^0 as the one that is transmitted explicitly. In order to bound the set of possible values for C_0^0, it is sufficient to satisfy the equations that bind the counters together and noting that the four unknown counters have to be non negative:

$$\begin{array}{lllll} C_0^0 \geq 0 & \Leftrightarrow & C_0^0 \geq 0, & C_1^0 \geq 0 & \Leftrightarrow \; C_0^0 \leq C_\bullet^0, \\ C_0^1 \geq 0 & \Leftrightarrow & C_0^0 \leq C_0^\bullet, & C_1^1 \geq 0 & \Leftrightarrow \; C_0^0 \geq C_\bullet^0 - C_1^\bullet; \end{array}$$

which can be summarized by: $\max(0, C_\bullet^0 - C_1^\bullet) \;\leq\; C_0^0 \;\leq\; \min(C_\bullet^0, C_0^\bullet)$. Given these bounds, the value of C_0^0 can be transmitted efficiently. Note that, when at least one of $C_\bullet^0$, $C_\bullet^1$, $C_0^\bullet$, or $C_1^\bullet$ is zero, then the lower and upper bounds become equal and C_0^0 can be determined without having to transmit any information.

The complete process of enumerating $\mathbf{D}$'s substrings consists in transmitting C_ϵ and C_0 in a special way and then by transmitting the counters of the longer substrings using butterflies, from length 2 to length N. When transmitting the counters of the L-bit substrings, each $(L-2)$-bit core is considered in turn, using a butterfly to help determine the counters of its four extensions. A core is considered only if, in doing so, there is *something new* that can be learned about its extensions. Section 4 introduce the tools that allows CSE to efficiently transmit the counters of all the substrings.

3.3 Sources of compression

At first glance, it might not be clear why the mere enumeration of $\mathbf{D}$'s substrings might lead to compression. However, there are multiple reasons why it does. First, the information contained in the enumeration of the substrings of length up to L bits constitutes a good summary of the information contained in the $(L+1)$-bit substrings.

C_{1w1}. The unknown counters are related by $C_{0w0} + C_{0w1} = C_{0w}$, $C_{1w0} + C_{1w1} = C_{1w}$, $C_{0w0} + C_{1w0} = C_{w0}$, and $C_{0w1} + C_{1w1} = C_{w1}$.

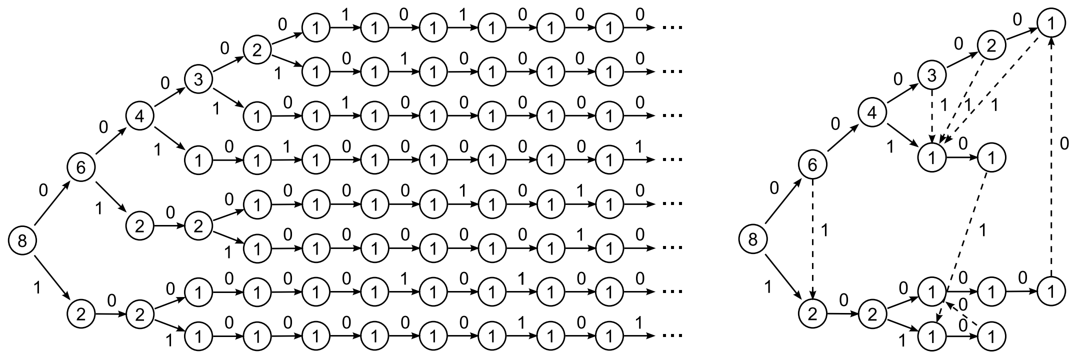


Figure 3: a) The IST for ‘01000001’; b) the corresponding CST

When enumerating the latter, there is no risk of being “caught by surprise”. Moreover, for the first values of L, the enumeration of the L-bit substrings requires fewer counts than that of the $(L+1)$-bit substrings. Second, the trees introduced in Section 4 filter the cores that matter and avoid sending the same information twice (or more). Third, in our experiments on the files of the Calgary Corpus, we were able to witness the effectiveness of the “summary effect” mentioned as the first reason. Here are two observations. Almost 78% of the butterflies that are processed are trivial ones, i.e. at least one of $C_{\bullet}^{0}$, $C_{\bullet}^{1}$, $C_{0}^{\bullet}$, and $C_{1}^{\bullet}$ is zero. Each butterfly provides the value of four counters and, in the case of trivial butterflies, we get these for free. Also, for at least 99% of the butterflies, the width of the range of possible values for C_{0}^{0} is 23 or less. The ranges are narrow because of the information that is known about the shorter substrings. Finally, compression can be improved by exploiting the fact that the values for C_{0}^{0} are not all equiprobable (as observed during our experiments).

4 Substring Trees

A structure that we use to enumerate the substrings is the *substring tree*. We introduce two versions of the substring trees: first, we present the *infinite* substring tree and, then, we present the *compacted* substring tree. The purpose of both trees is to indicate the number of occurrences of any substring of the original data $\mathbf{D}$.

4.1 Infinite substring trees

The *infinite substring tree with null counts* (IST0) for $\mathbf{D}$ is defined as follows. For each $w \in \{0,1\}^*$, there exists a node n_w. We say that w is the *address* of n_w. Node n_w is labeled with C_w, the number of occurrences of w in $\mathbf{D}$. Each arc is labeled with a bit. For each pair of nodes n_w and $n_{w\,b}$, there exists an arc from n_w to $n_{w\,b}$ that is labeled with b. We say that $n_{w\,b}$ is the b-child of n_w. Note that the counter of a node is the sum of the counters of its children. Note also that the sequence of the bits that label the arcs on the path from the root to a node n is equal to n's address.

Next, we define the *infinite substring tree* (IST) for $\mathbf{D}$ using the IST0 for $\mathbf{D}$. The IST is the sub-graph of the IST0 in which we keep only the nodes labeled with strictly positive labels and the arcs that connect these nodes. Note that, by construction, each node in an IST has either one or two children. Note also that there exists a one-to-one correspondence between the IST0s and the ISTs. Figure 3a shows the IST of our example original data ‘01000001’.

Note that the shape and the node labels of an IST from level 0 down to level N essentially correspond to the enumeration of the substrings of lengths between 0 and N. The transmission of the node labels of the IST, level by level from the root down to level N, would be a complete enumeration of $\mathbf{D}$'s substrings.

In the presentation of the compacted substring trees below, we use a relation of order between infinite sub-trees. First, for IST0s, we say that t is *larger* than t', denoted $t \sqsupseteq t'$, if, for all $w \in \{0,1\}^*$, $C_w \geq C'_w$, where C_w and C'_w are the counters that label the nodes at address w of t and t', resp. Second, for ISTs, t is *larger* than t' if the IST0 corresponding to t is larger than the IST0 corresponding to t'.

We state three properties of ISTs. *Prop. 1.* If there exists a node n_{uw} at address uw, then there also exists a node n_w at address w and the respective corresponding sub-trees t_{uw} and t_w are such that $t_w \sqsupseteq t_{uw}$. *Prop. 2.* Let t_v and t_w be two sub-trees of the IST and let C_v and C_w be the counters at their respective roots. If $t_v \sqsupseteq t_w$ and $C_v = C_w$, then $t_v = t_w$. *Prop. 3.* Let t_v and t_w be sub-trees rooted at n_v and n_w, respectively. If $t_v = t_w$, then the shorter of v and w is a suffix of the other.

4.2 Compacted substring trees

The *compacted substring tree* (CST) for $\mathbf{D}$ is a more practical representation of the IST for $\mathbf{D}$. In fact, it is not a tree but rather a graph. The CST is a finite graph that is *isomorphic* to the IST. By "isomorphic", we mean that there is a mapping from the nodes of the IST to the nodes of the CST that preserves the connectivity, the counters that label the nodes, and the bits that label the arcs.

Here we are interested in the *smallest* graph that is isomorphic to $\mathbf{D}$'s IST. Let T be $\mathbf{D}$'s IST. Note that there exists at least one finite graph that is isomorphic to T.[4] Since there exists at least one such graph, then the set of such graphs is non empty and we can find one that has the minimum number of nodes.

Note that, even if the CST is not a tree, we prefer to call it a tree since it is isomorphic to (an infinite) one and that we intend to traverse it as if it were a tree. Still, for the sake of the presentation in this paper and for the implementation, we distinguish the arcs that go "forward" from a level-L node to a genuine level-$(L+1)$ one from those that go "back" from a level-L node to a level-L' one, where $L' \leq L$. In the picture presented in the following example, arcs that go forward are depicted with solid lines, as in Figure 3a, while arcs that go backward are depicted with dashed lines. Figure 3b presents the CST for our example original data '`01000001`'.

In our experiments, we observed that the number of nodes in the CST was consistently $2N-1$.[5] Note that this is also the case in our example's CST. We conjecture that the CST for (a non-repetitive) $\mathbf{D}$ always has $2N-1$ nodes.

Note that the CST can be implemented as a data structure whose size remains realistic in practice. Moreover, given that the CST for $\mathbf{D}$ has a linear number of nodes, then the time it takes to traverse every node once is linear.

[4]We can easily build one. First, observe that all nodes at level N have counters equal to 1. Next, let n_0 be the node at address $\mathbf{D}$. We modify the level-$(2N-1)$ descendant of n_0 so that its child is now n_0. Note that, by doing so, we create a cycle. Note also that the N-arc cycle is labeled with $\mathbf{D}$. Next, we modify each of the level-N nodes, except n_0, so that its child is now the appropriate node in the cycle. Finally, we remove all the nodes that are unreachable from the root.

[5]CSTs with fewer nodes would be possible if we allowed $\mathbf{D}$ to be repetitive. If $\mathbf{D} = w^k$, for $k > 1$, then the CST for $\mathbf{D}$ is identical to the one for w except that all counters are multiplied by k.

One of the effects of having a finite graph is that nodes do not have unique addresses anymore. For instance, in Figure 3b, the node at address '`000`' has a single address but the node at address '`1`' also has address '`01`' and the node at address '`100`' has an infinite number of addresses since it is part of a cycle.

If a node n has two addresses w and w', then one address has to be the suffix of the other. A corollary of this statement is that every node has a unique shortest address. From now on, when we refer to *the* address of a node, we refer to the shortest of all the node's addresses, unless stated otherwise.

During a level-by-level construction of a CST, we need to detect if a given child node that we are about to create is isomorphic to some node that appears at a higher level.[6] Let n_w be that would-be node. Address w might not be n_w's shortest one, but we do not know that yet. Now, let n_x be such that there exists b such that $b\,x = w$. Then we have that there exists a node m located on a higher level that is isomorphic to n_w if and only if $C_x = C_w$. When $C_x = C_w$, n_x is isomorphic to n_w and we might as well choose $m = n_x$. During the construction of the CST, if $C_x = C_w$, then n_w must not be built and, instead, a backward arc should be added from its parent to n_x. Otherwise, n_w must be built. Locating node n_x from node n_w's parent can be done in constant time if we add *suffix links* to the implementation of the CST.

4.3 Back to the trace of the enumeration

We complement the trace that is presented in Subsection 3.1 with the issue of the incremental construction of the CST by the decompressor. First, root node n_ϵ must invariably be constructed with $C_\epsilon = 8$. Next, we consider the nodes on level 1. Once the decompressor learns that $C_{\mathtt{0}} = 6$ and $C_{\mathtt{1}} = 2$, it knows that both nodes $n_{\mathtt{0}}$ and $n_{\mathtt{1}}$ exist but these might happen to be isomorphic to a node that appears on a higher level, namely the root n_ϵ. By verifying that the criterion $C_\epsilon > C_{\mathtt{0}}$ holds, the decompressor determines that $n_{\mathtt{0}}$ is not isomorphic to n_ϵ. By verifying that $C_\epsilon > C_{\mathtt{1}}$, it draws the same conclusion for $n_{\mathtt{1}}$. Both level-1 nodes must be created. Finally, on processing level 2, the decompressor learns that $C_{\mathtt{00}} = 4$, $C_{\mathtt{01}} = 2$, $C_{\mathtt{10}} = 2$, and $C_{\mathtt{11}} = 0$. It immediately concludes that nodes $n_{\mathtt{00}}$, $n_{\mathtt{01}}$, and $n_{\mathtt{10}}$ exist but not $n_{\mathtt{11}}$. Next, given that $C_{\mathtt{0}} > C_{\mathtt{00}}$ and $C_{\mathtt{0}} > C_{\mathtt{10}}$, the decompressor creates nodes $n_{\mathtt{00}}$ and $n_{\mathtt{10}}$. On the other hand, given that $C_{\mathtt{1}} = C_{\mathtt{01}}$, the decompressor *does not* create $n_{\mathtt{01}}$ but instead adds a backward arc labeled with 1 from $n_{\mathtt{0}}$ to $n_{\mathtt{1}}$.

5 Summary of the CSE Technique

Here we give a pretty terse overview of the complete CSE technique. A file (or a stream) is compressed by dividing it into blocks and compressing each block. $\mathbf{D}$ is taken to be each of the successive blocks.

The first step consists in determining if $\mathbf{D}$ is repetitive. This can be done in linear time by looking for the longest sequence(s) of `0`s. If it is unique, then $\mathbf{D}$ is non-repetitive.[7] Otherwise, let k be its number of occurrences and L be its length. We then compare the bits that follow the sequences of `0`s. If, after having compared the remaining $N - kL$ bits on total, we cannot single out a unique smallest substring, then $\mathbf{D}$ is made of k' repetitions, for $1 < k' \leq k$, otherwise $\mathbf{D}$ is non-repetitive.

[6]Here, we make a slight abuse of language. When we say that a node n_u is isomorphic to another node n_v, we mean that the IST sub-trees that are rooted at n_u and n_v are isomorphic.

[7]The special case where $\mathbf{D}$ is made only of `0`s or `1`s can easily be handled.

File	Gzip	BWT	PPM	$\overline{\text{Btf}}$	Btf	BTF
`bib`	2.51	2.07	**1.91**	2.54	2.56	1.98
`book1`	3.25	2.49	2.40	3.14	3.06	**2.27**
`book2`	2.70	2.13	2.02	2.74	2.72	**1.98**
`geo`	5.34	**4.45**	4.83	6.03	5.52	5.35
`news`	3.06	2.59	**2.42**	3.33	3.32	2.52
`obj1`	**3.84**	3.98	4.00	5.10	4.46	4.46
`obj2`	2.63	2.64	**2.43**	3.03	3.02	2.71
`paper1`	2.79	2.55	**2.37**	2.79	2.80	2.54
`paper2`	2.89	2.51	**2.36**	2.77	2.77	2.41
`paper3`	3.11	—	—	2.95	2.96	**2.73**
`paper4`	3.33	—	—	**3.17**	3.20	3.20
`paper5`	3.34	—	—	**3.29**	3.33	3.33
`paper6`	2.77	—	—	2.75	2.76	**2.65**
`pic`	0.82	0.83	0.85	2.05	0.79	**0.77**
`progc`	2.68	2.58	**2.40**	2.76	2.77	2.60
`progl`	1.80	1.80	**1.67**	1.90	1.89	1.71
`progp`	1.81	1.79	**1.62**	1.99	1.96	1.78
`trans`	1.61	1.57	**1.45**	2.16	2.07	1.60

Figure 4: Experimental results

When $\mathbf{D}$ is repetitive, the fact is signaled to the decompressor, along with k', and the compression is performed on the $(|\mathbf{D}|/k')$-bit factor.

The second step consists in first building a suffix tree for $\mathbf{D}^2$ using a linear-time technique [6]. $\mathbf{D}$ is duplicated in order to simulate circularity. Then, counts are inserted in the suffix tree at depth N. A count of 1 is used for any N-bit suffix that has not yet reached the end of $\mathbf{D}^2$, otherwise, a count of 0 is used. Next, the counts are summed in a bottom-up fashion to reproduce those that would appear in an IST. Finally, the CST is built by performing synchronous breadth-first traversals of both the suffix tree and the CST that is being constructed. Isomorphism tests ensure that the constructed CST does not contain duplicate nodes for isomorphic sub-trees.

The third step consists in transmitting C_ϵ first, then C_0 subject to $0 \leq C_0 \leq C_\epsilon$, and finally, for L going from 2 to N, and for each $(L-2)$-bit core w such that at least one of n_{0w} and n_{1w} truly lies on level $L-1$, in processing the butterfly of core w.

The fourth step consists in sending the rank of $\mathbf{D}$, i.e. the number of rotations that are lexicographically smaller than $\mathbf{D}$. Note that the rank lies between 0 and $N-1$, inclusively. The rank is obtained by going down the CST following the path $\mathbf{D}$ and adding C_{w0} each time a transition is made from n_w to n_{w1}. The total is $\mathbf{D}$'s rank.

Note that the decompressor has to participate in the third and fourth steps and also in the first step when the block is repetitive. It performs the construction of the CST during the third step. All the described operations can be performed in time that is linear in the size of the block, thus in linear time for a complete file.

6 Experimental Results

We have implemented a prototype that performs CSE. It first divides files that are too large into blocks. Given $\mathbf{D}$, instead of building the CST in the way described in Section 5, it uses a sorting technique that is reminiscent of the Bucket Sort [3] and builds the CST by extracting counts from the sorted list of rotations. This causes our prototype not to run in linear time. Next, it transmits the counters of the CST using arithmetic coding [8]. Finally, it transmits the rank of the N-bit substring that is $\mathbf{D}$.

We present results for three variants of our prototype: $\overline{\text{Btf}}$, Btf, and BTF. Given a butterfly, $\overline{\text{Btf}}$ predicts the value of C^0_0 by assigning equal probabilities to all the possible values of C^0_0, from the lower to the upper bound. Such a prediction is pretty naïve. For example, if $C^0_\bullet = C^\bullet_0 = 100$ and $C^1_\bullet = C^\bullet_1 = 1$, we have that $99 \leq C^0_0 \leq 100$ and $\overline{\text{Btf}}$ considers that $P(C^0_0 = 99) = \frac{1}{2} = P(C^0_0 = 100)$. However, intuition suggests that $C^0_0 = 99$ is much more probable. Unlike $\overline{\text{Btf}}$, Btf and BTF adaptively encode the values of C^0_0, using the *kinds* of butterflies as contexts (that is, specific ranges for $C^0_\bullet$, $C^1_\bullet$, $C^\bullet_0$, and $C^\bullet_1$). Both $\overline{\text{Btf}}$ and Btf use blocks of 32 kB while BTF uses blocks of 1MB, which allows each benchmark file to be processed without being divided.

We ran our prototype on the files of the Calgary Corpus [7]. We compared the compression efficiency of the prototype with that of `gzip` [5] set at maximal compression and that reported for PPM*C and the Burrows-Wheeler transform (BWT) in [2]. The measurements (in bits per character) are presented in Figure 4.

CSE outperforms `gzip` and BWT too on most files and it is not so far behind PPM*C. We see that the adaptive encoding of C^0_0 helps but it is when coupled with the use of large blocks that our technique becomes truly competitive. BTF still seems to have some difficulty with binary data like machine code. We suspect that the initial conversion from bytes to bits causes our technique to lose track of the original byte boundaries, which might not be easily recovered in binary data and incur a penalty in prediction performance. Note that, by using 32 kB blocks, $\overline{\text{Btf}}$ and Btf are compared unfairly to the other techniques, which allow themselves to have a wider view on the data.[8] Note also that CSE, like PPM*C and BWT, is able to compensate for its inability to identify *matches*, which `gzip` effectively exploits. Finally, note that CSE, in its C^0_0 prediction technique, does not have the advantage of enjoying the same (considerable) amount of work that has been invested on PPM's escape mechanisms.

The experiments were run on a 3 GB, 2.83GHz, Intel QuadCore machine. Compressing and decompressing all the files of the Calgary Corpus required less than 16% of the computer's memory and took less than 2 minutes. It means that CSE attains its compression effectiveness while only requiring practical amounts of resources, in time and space.

7 Discussion

CSE has some links to *Prediction by Partial Matching* (PPM) [2]. In some sense, the IST nodes that are located at addresses of length $k+1$ and their associated counters can be viewed as the equivalent of an order-k model used by PPM. An important difference is that CSE's "models" are *exact* while PPM's are approximate as nothing is known for sure about the yet unseen part of the data. CSE avoids the need for *escape characters*. When a level of the CST is transmitted, the operation can be seen as the description of the order-$(k+1)$ models based on the order-k models. Our technique does not try to predict $\mathbf{D}$'s bits. Instead, it progressively describes $\mathbf{D}$ using ever more powerful models until a perfect knowledge of $\mathbf{D}$'s contents is gathered. Finally, CSE is not constrained to make predictions in a left-to-right manner.

CSE also has links to the compression techniques based on *anti-dictionaries* [4]. To any forbidden word in an anti-dictionary corresponds a null counter in CSE.

CSE benefits from the presence of frequently occurring words, in a way that is somehow analogous to the dictionary-based techniques LZ77 [9] and LZ78 [10]. For example, frequent occurrences of "the" in English lead LZ77 to identify matches that include "the" and LZ78 to introduce "the" and its extensions in its dictionary. CSE will tend to predict the appearance of the bits at one end of "the" when presented with the bits of the other end.

[8] Technically, `gzip` cannot see more than 32 kB at a time but it does allow itself to gather statistics on much more than 32 kB of original data.

Finally, like the *Burrows-Wheeler Transform* (BWT) [1], CSE has to work on a block-by-block basis. The larger the blocks are, the better the compression is. Also, like BWT, the last item that the compressor transmits to the decompressor is the rank of the rotation that is identical to the original data.

8 Future Work

Many things should be tried in order to improve this work. First, we must improve the way C^0_0's value is predicted. A flat probability distribution for the possible values of C^0_0—like $\overline{\text{Btf}}$ does—is clearly unrealistic and simply learning the distribution for C^0_0—like Btf and BTF do—incurs a penalty for the learning process. Second, it is necessary to demonstrate that a CST always has size $2N - 1$. The linearity in time and space of our technique is based on that. Third, it would be interesting to formally demonstrate that the technique is universal. A prerequisite is the development of the "right" way to predict C^0_0. We expect to have multiple cases to consider in order to get a demonstration of universality for a finite-order Markov source. Fourth, an empirical study that ought to be conducted consists in measuring the impact of the loss of the phase of the bits inside of their bytes. By that, we refer to the possible cause for the inferior performance on binary data that we mention in Section 6. Finally, we intend to determine how to exploit the fact that there exists a Hamiltonian cycle running through the set of L-bit substrings, and whether substantial savings can be extracted from this fact.

Acknowledgements

This work was supported by the Natural Science and Engineering Research Council of Canada. We wish to thank Hidetoshi Yokoo and the anonymous referees for their useful comments.

References

[1] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[2] J. G. Cleary and W. J. Teahan. Unbounded length contexts for PPM. *The Computer Journal*, 40(2/3):67–75, 1997.

[3] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2001.

[4] M. Crochemore and G. Navarro. Improved antidictionary based compression. In *Proc. of the International Conf. of the Chilean Computer Science Society*, pages 7–13, 2002.

[5] J. L. Gailly and M. Adler. The GZIP compressor. `http://www.gzip.org`.

[6] E. Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14:249–260, 1995.

[7] I. Witten, T. Bell, and J. Cleary. The Calgary corpus, 1987. `ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus`.

[8] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Comm. of the ACM*, 30(6):520–540, 1987.

[9] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory*, 23(3):337–342, 1977.

[10] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. on Information Theory*, 24(5):530–536, September 1978.