Using Synchronization Bits to Boost Compression by Substring Enumeration

Danny Dubé
Université Laval, Canada
Email: Danny.Dube@ift.ulaval.ca

***Abstract*—A new lossless data compression technique called compression via substring enumeration (CSE) has recently been introduced. It has been observed that CSE achieves lower performance on binary data. An hypothesis has been formulated that suggests that CSE loses track of the position of the bits relative to the byte boundaries more easily in binary data and that this confusion incurs a penalty for CSE. This paper questions the validity of the hypothesis and proposes a simple technique to reduce the penalty, in case the hypothesis is correct. The technique consists in adding a preprocessing step that inserts synchronization bits in the data in order to boost the performance of CSE. Experiments provide strong evidence that the formulated hypothesis is true and they demonstrate the effectiveness of the use of synchronization bits.**

I. Introduction

Recently, a new lossless data compression technique called compression via substring enumeration (CSE) has been introduced [1]. CSE compresses data by sending the number of occurrences of every distinct substring of the data, enumerating the substrings from the shortest to the longest. While the first experiments do not demonstrate that CSE is the best data compression technique yet, they show that it is already competitive and that it is a very promising technique.

However, it has been noted that the performance of CSE tends to be lower when it deals with binary data as opposed to text-like data. An hypothesis has been formulated that suggests that CSE, since it works at the bit level, is unaware of the position of the substrings it manipulates with respect to the byte boundaries. In other words, CSE is unaware of the *phase* of the substrings. Indeed, CSE considers the data to be just a string of bits and the substrings may start and end at arbitrary bit positions. This bit-oriented point of view would happen to be a disadvantage for CSE since the statistics of the bits need not be the same at every position inside of the bytes. Still, by enumerating ever longer substrings, CSE manipulates sets of substrings that would tend to be in the same phase. In the case of binary data, there would be fewer clues in the original bytes that would help CSE to separate substrings of different phases and, consequently, CSE would have to make more predictions on sets of mixed-phased substrings. Mixed-phased substrings, when considered together, would have blended statistics which would be more poorly predicted by CSE, leading to worse compression. At least, this explanation is that of the given hypothesis. We intend to test the validity of this hypothesis and see whether we can alleviate the alleged problem of the phase of the substrings.

In Section II, we review the basics of CSE and we present in more details what the phase problem is supposed to be. In Section III, we propose a simple method that attempts to lessen the penalty incurred by losing track of the phase. This method simply consists in adding *synchronization bits* to the original data to help CSE take the byte boundaries into account. The effects of the proposed method are experimentally measured in Section IV, using a variety of synchronization schemes. Section V concludes the paper and mentions future work on the phase problem and, more generally, on CSE.

II. Compression via Substring Enumeration

A. Review of the Technique

CSE manipulates the data to compress as a string of bits. We denote the data by $\mathbf{D} \in \{0,1\}^+$ and the length of the data by $N = |\mathbf{D}|$. CSE's compression essentially proceeds like the following two embedded **for** loops:

For l **:=** 1 **to** N **do**
  For every distinct l-bit substring w of $\mathbf{D}$ **do**
    Send number of occurrences of w in $\mathbf{D}$

The data is considered to be circular and, as such, it is possible to have substrings that wrap around $\mathbf{D}$. We describe the operations performed by CSE using a small example. Let $\mathbf{D} =$ `01000001`. Figure 1 illustrates the enumeration of the substrings that is performed when compressing $\mathbf{D}$. Note that, for a given length, the substrings are enumerated in lexicographic order, not in the order in which they appear in $\mathbf{D}$. It is the number of occurrences of a substring (e.g., the '4' in '4×`00`') and not the substring itself that needs to be sent at each step.

This description of CSE looks wasteful as, except for a very repetitive $\mathbf{D}$, there are apparently $\theta(N^2)$ numbers that must be sent in order to describe $\mathbf{D}$. However, a practical data structure is used by CSE and only $O(N)$ numbers need be sent, resulting in an $O(N)$ time and space complexity for CSE [1]. The original paper only includes a conjecture stating that the used data structure does have linear size but the conjecture has been proved independently and the proof has yet to appear [2].

The original paper mentions many reasons why CSE is able to effectively compress data [1]. In our opinion, the most important reason is that the enumeration of the substrings of a particular length provides a lot of information about the enumeration of the substrings that are one bit longer. CSE takes

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

Fig. 1. Substring enumeration for '01000001'.

into account the fact that a substring $a\,w\,b$, where $a,\, b \in \{0,1\}$ and $w \in \{0,1\}^*$, is an extension of both substrings $a\,w$ and $w\,b$. The enumeration proceeds by predicting the number of occurrences of the l-bit substrings $0\,w\,0$, $0\,w\,1$, $1\,w\,0$, and $1\,w\,1$ at once, which all share a common $(l-2)$-bit *core* w. CSE takes the equations

$$C_v = C_{v\,0} + C_{v\,1} \quad \text{and} \quad C_v = C_{0\,v} + C_{1\,v}$$

into account when making its prediction, where C_u denotes the number of occurrences of a substring u in $\mathbf{D}$.

CSE has some (more or less tight) links to previous compression techniques, namely to prediction by partial matching [3], [4], antidictionaries [5], LZ77 [6], LZ78 [7], and the Burrows-Wheeler transform [8].

B. The Problem with the Phase

In the experiments presented in the original paper [1], it has been observed that CSE is not as competitive on binary (or non-text) data as on text-like data. The authors posed the hypothesis stating that CSE, viewing files as strings of bits, suffers from unawareness of the *phase*, i.e. the position of the bits of the substrings with respect to the boundaries of the bytes. Indeed, all the benchmark files are made of bytes and the CSE prototype abstracts away the concept of bytes and views the files as eight-times longer strings of bits. Let us make an illustration of the difficulties caused by the unawareness of the phase and let us consider the case of executable code. Suppose that we have the following two substrings with the same core:

$$\begin{array}{cccccccccccc} a_2 & 0_3 & 1_4 & 1_5 & 1_6 & 0_7 & 0_8 & 1_1 & 0_2 & 0_3 & b_4 \\ c_7 & 0_8 & 1_1 & 1_2 & 1_3 & 0_4 & 0_5 & 1_6 & 0_7 & 0_8 & d_1 \end{array}$$

where the indices indicate the phase of the bits. Note that CSE is *unaware* of those indices. Given that these substrings come from executable code, it might well be the case that both substrings start in bytes that encode the opcodes of machine instructions. Then, in such a case, bit a could more likely be part of the encoding of the operation performed by an instruction while bit c could more likely be part of the encoding of a register number used by another instruction. As such, bit a needs not necessarily follow the same statistical distribution as bit c. A similar phenomenon is to be expected for bits b and d. Now, note that CSE is unaware of the phase in *all* data, not just in executable code. So why is it hypothesized that binary data is especially problematic for CSE? It is because text-like data would have the tendency to include "clues" about the phase. For instance, (mostly) pure ASCII text is such that (almost) all the bytes have their most significant bit at zero. These zero bits tend to group same-phase substrings together. For example, if we consider two 8-bit out-of-phase substrings each with their phase-1 bit at zero:

$$\begin{array}{cccccccc} a_6 & a_7 & a_8 & 0_1 & a_2 & a_3 & a_4 & a_5 \\ b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & 0_1 & b_2 \end{array}$$

and if we suppose that the other bits are purely random, then these substrings have half the chances of being equal due to the misalignment of the phase-1 zero bits. The longer the considered substrings, the more the phase-1 zero bits reduce the chances that the substrings are equal. Still, the zero bits offer *no guarantee* that two substrings need to be in the same phase, no matter how long the latter are. Nevertheless, the hypothesis is that the mere tendency to have phases matched for the substrings in text-like data helps CSE to better compress text-like data. Note that the hypothesis does not claim that CSE is made *more aware* of the phase in any way, it just claims that same-phase substrings tend to group together when their length increases. In the case of binary data, there typically is no such bit that is constant in every byte. There might still exist clues that suggest the phase but longer substrings need to be manipulated before they start to share phase. Consequently, CSE makes poorer predictions on a larger fraction of the substrings.

Making CSE aware of the phase or making sure that same-phase substrings tend to (or are forced to) group together, no matter the kind of data that is being compressed, could lead to improvements. This idea is the one that is proposed in the next section.

III. Using Synchronization Bits

What we propose is to make CSE take into account, in a way or another, the phase information. One could think of countless ways to reach that goal. Here, we intend to study the effects of a particularly simple scheme: adding a preprocessing step that inserts *synchronization bits* in the data. The simplicity lies in the fact that the original CSE technique needs not be modified in any way. Moreover, the insertion (and removal) of the synchronization bits is a very simple and cheap operation. Denoting the CSE compressor by c, the decompressor, by d, the synchronization bit adder, by s, and the synchronization bit remover, by r, then the proposed method uses $c \circ s$ as compressor and $r \circ d$ as decompressor. Since the original CSE

technique is lossless and the combined operation of inserting and removing the synchronization bits is also lossless, then the proposed method remains lossless. More formally, since both $d \circ c$ and $r \circ s$ are the identity function, then $r \circ d \circ c \circ s$ is also the identity function.

A. Synchronization Schemes

There is a large amount of work on synchronization codes [9], [10], [11], [12]. In this work, we are not necessarily interested in the state of the art on synchronization codes. Rather, we only intend to use one of the simplest synchronization schemes. Since we are interested in nothing more than making the position of the bits inside of the bytes explicit, we choose to insert synchronization bits on a per-byte basis. More specifically, we are only interested in the insertion of fixed bit paddings inside of the bytes. All of the synchronization schemes that we consider can be characterized by 9 bit strings, $w_1, w_2, \ldots, w_9 \in \{0,1\}^*$, and the following map M on the bytes:

$$M(b_1\, b_2\, \ldots\, b_8) = w_1\, b_1\, w_2\, b_2\, \ldots\, w_8\, b_8\, w_9,$$

where $b_1 \ldots b_8$ are the 8 bits that form a byte. We say that a particular scheme inserts k bits per byte if $|w_1 \ldots w_9| = k$. We also say that it is a k-bit synchronization scheme.

Arguably, the synchronization schemes that we choose to consider are very simple. There exist more sophisticated synchronization codes whose "performance" are superior. However, the deliberate simplicity of the considered schemes becomes an advantage in this work. There are two reasons for this. First, a simple synchronization scheme obviously leads to a simple implementation. Second, we must keep in mind that the bit insertion is a preprocessing step and that the data in which the synchronization bits are inserted is fed to the CSE compressor. The compressor must now compress the original data *plus* the synchronization bits. In that respect, we want to avoid any sophistication in the synchronization scheme to make sure that the patterns of the synchronization bits remain easy for the compressor to *learn*.

Note that our proposal of inserting synchronization bits is a kind of gamble. Indeed, when we use a k-bit scheme, we cause an expansion of the original file in the hope that the resulting file can be turned into a compressed file that is even smaller than when the original file itself is compressed. That is, given $\mathbf{D}$, we turn it into $s(\mathbf{D})$, which is $\frac{k+8}{8}$ times larger than $\mathbf{D}$, in the hope that

$$|c(s(\mathbf{D}))| < |c(\mathbf{D})|.$$

This is stronger than asking for $s(\mathbf{D})$ to be more compressible (in terms of compression ratio) than $\mathbf{D}$, i.e.

$$\frac{|c(s(\mathbf{D}))|}{|s(\mathbf{D})|} < \frac{|c(\mathbf{D})|}{|\mathbf{D}|},$$

which is trivial to achieve.

B. Reliable Synchronization

For any non-trivial synchronization scheme (i.e. one for which $|w_1 \ldots w_9| > 0$), the inserted bits become clues for CSE to gather same-phase substrings when the latter get long enough. However, while all schemes provide clues about the phase, certain schemes are more informative than mere clues. We say that a scheme provides *reliable* synchronization when, for any two sufficiently long substrings, the latter can be equal only if they share the phase. We say that a scheme itself is *reliable* if it provides reliable synchronization. Given that we are interested into synchronization with the byte boundaries and that a k-bit scheme transforms any byte into $k+8$ bits, the *reliability* criterion that we choose requires that two substrings of at least $k+8$ bits be different whenever they are on different phases.[1]

Here is an example of a non-reliable scheme: the scheme that inserts a '1' bit at each end of the byte and a '0' bit between an even-phased bit and an odd-phased bit, when the latter appear in that order. More precisely, the scheme is characterized by $w_1 = w_9 = 1$, $w_3 = w_5 = w_7 = 0$, and $w_2 = w_4 = w_6 = w_8 = \epsilon$. The fact that the scheme is not reliable is demonstrated by the following two 13-bit substrings:

1 0 0 0 1 1 0 0 0 0 0 0 1
1 0 0 0 1 1 0 0 0 0 0 0 1

in which the original bits have been underlined. The first substring is directly $M(00110000)$. The second substring is on a different phase: it starts with the 5 last bits of a transformed byte and continues with the 8 first bits of the next transformed byte. The substrings are clearly on different phases and they are long enough ($5 + 8$ bits). Yet, they are equal.

Now, here is an example of a reliable scheme. It is used in the experiments that are presented in the next section. It adds a '0' after the 6th bit and '0111' after the last. Figure 2 shows that any two 13-bit substrings (for the 5 inserted bits) that are on different phases must be different.[2] The underscores denote arbitrary original bits. The values of the latter do not matter since the synchronization bits are sufficient to produce bit mismatches.[3]

C. Learnability of the Synchronization Schemes

We mention above that we consider only simple synchronization schemes because these are easy to learn for CSE. We

[1] In other words, once a reliable k-bit scheme has been applied, the fact that two substrings are at least $(k+8)$-bits long and that they are on different phases implies that the substrings are necessarily different. Naturally, the reverse does not hold: even if two substrings of at least $k+8$ bits are on the same phase does not mean that they are equal.

[2] In other words, by considering any two distinct rows i and i' ($i \neq i'$) of the matrix, one can always find a column j such that the entries (i, j) and (i', j) contain synchronization bits of opposite values.

[3] Since every row of the matrix is a rotation of the first one, it is sufficient to show that the first row is necessarily different from each row i', for $i' > 1$. In each row i', the bit that causes a mismatch with the first row is typeset in bold.

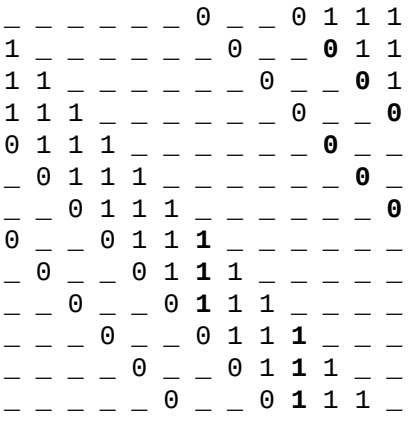

Fig. 2. Demonstration of the reliability of a 5-bit scheme.

explain here what it means for CSE to *learn* the synchronization schemes. In its enumeration of the substrings from the shortest to the longest, CSE quickly reaches lengths that are great enough to have the considered sets of substrings to be necessarily synchronized. Whenever CSE manipulates a set of synchronized substrings and that the bits to predict at either end of the core happen to be synchronization bits, then the prediction is made "for free", i.e. without any need for an explicit transmission of information from the compressor to the decompressor. We illustrate the process of a prediction step in more details in the following.

For the sake of illustration, we consider the reliable synchronization scheme of Figure 2. We use phases, or bit positions, that range between 1 and 13. For instance, '0' synchronization bits appear at phases 7 and 10 and '1' synchronization bits appear at phases 11, 12, and 13. Let us consider the substrings of length, say, 23 bits that have w as their core. Let S be the multiset of the occurrences of these substrings: $\{a_1\, w\, b_1,\ \ldots,\ a_k\, w\, b_k\}$.[4] At this step, we need to predict the numbers of occurrences $C_{0\,w\,0}$, $C_{0\,w\,1}$, $C_{1\,w\,0}$, and $C_{1\,w\,1}$, given that the numbers of occurrences $C_{w\,0}$, $C_{w\,1}$, $C_{0\,w}$, $C_{1\,w}$, and C_w are already known. Note that we have $C_w = k$. With its 21 bits, w is long enough to have all of its occurrences synchronized; likewise for the substrings in S. Now, we consider two cases: one where the leading bits of the substrings appear at phase 12 and one where the leading bits appear at phase 9.

Let us consider the case where the leading bits of the substrings in S appear at phase 12. Note that, in this case, the leading bits happen to be synchronization bits. Since the bits appearing at phase 12 are '1', then we have that $a_1 = \ldots = a_k = 1$. In other words, all the occurrences of w are preceded by '1' (and none by '0'). This implies that the equalities $C_{0\,w} = 0$ and $C_{1\,w} = k$ must have been established during the enumeration of the 22-bit substrings. The fact that $C_{0\,w} = 0$ implies that $C_{0\,w\,0} = 0$ and $C_{0\,w\,1} = 0$. The latter equalities imply that $C_{1\,w\,0} = C_{w\,0}$ and $C_{1\,w\,1} = C_{w\,1}$. Both the compressor and the decompressor can reach the same conclusions without the need for any information to be transmitted from the former to the latter. These necessary conclusions follow from the fact that $C_{0\,w} = 0$.[5] More related to the matter of synchronization, such necessary conclusions follow whenever the leading (or trailing) bits of the substrings in S happen to be synchronization bits.

Let us consider the case where the leading bits of the substrings in S appear at phase 9. In this case, the leading bits are *not* synchronization bits. But what about the trailing bits? They appear at phase 5 (i.e. 22 positions more to the right, modulo 13), so they are not synchronization bits either. In other words, both leading and trailing bits are original bits. Consequently, there is no synchronization effect that forces at least one of $C_{w\,0}$, $C_{w\,1}$, $C_{0\,w}$, and $C_{1\,w}$ to be zero. Note that there is nothing that prevents the four numbers to be zero either; it all depends on the bits originally present in $\mathbf{D}$. In general, such a prediction step might require the transmission of information from the compressor to the decompressor.

A transmission "for free" during an enumeration step happens each time CSE manipulates a multiset S where the core w is long enough to be synchronized and the phase of either the a_is or the b_is happens to be that of synchronization bits. In many other circumstances, the transmission needs not be "for free": when none of the a_is or the b_is have the phase of synchronization bits; when w is too short to be synchronized; or when the synchronization scheme itself is not reliable in the first place. In the case of a reliable synchronization scheme, we say that CSE pays a cost for *learning* the synchronization scheme only during the enumeration steps in which the core is short. Once the manipulated substrings are long enough, the description of the synchronization bits ceases to incur any cost. In the case of non-reliable synchronization schemes, the picture is less clear.

IV. Experiments

In order to test the validity of the hypothesis about the unknown-phase penalty incurred by CSE and also in order to measure the effectiveness of our proposed technique, we make some experimental comparisons. We measure the performance of CSE when preprocessing the data using k-bit schemes, for various values of k. We also include measurements on CSE without any synchronization scheme and measurements on the Burrows-Wheeler transform (BWT) [8], a variant of prediction by partial matching called PPM*C [3], [4], and a technique based on antidictionaries [5]. The measurements for both BWT and PPM*C come from the paper that presents PPM*C [4].

The variant of CSE that we use in these experiments is similar to the one that is the most competitive in the original presentation of CSE [1]. It is a variant that learns how to make predictions on the numbers of occurrences. However,

[4]We use a *multiset* instead of a *set* because we are interested in counting the occurrences, not just in detecting their existence. A set would have at most four elements since a_i, $b_i \in \{0, 1\}$, for $1 \leq i \leq k$. A multiset is a data structure with the right degree of precision since, for the sake of the prediction that is about to happen, we need to know the number of occurrences of the substrings but not their relative order in $\mathbf{D}$.

[5]A similar effect would happen in case $C_{1\,w} = 0$, $C_{w\,0} = 0$, or $C_{w\,1} = 0$.

k	Synchronization Scheme
1	_ _ _ _ _ _ _ _ 0
2	_ _ _ _ _ _ _ _ 0 1
3	_ _ _ _ _ _ _ _ 0 1 1
4	_ _ _ _ _ _ _ _ 0 1 1 1
5	_ _ _ _ _ _ 0 _ _ 0 1 1 1

Fig. 3. Synchronization schemes used in the experiments.

the variant that we use here processes blocks of 512 KB at a time (i.e. 2^{22} bits per block), instead of 1 MB. Some of the benchmark files that we use do not fit in a single block. Others become larger than a block once synchronization bits are inserted. Dividing a file into multiple blocks reduces the effectiveness of CSE. The set of benchmark files that we use is the Calgary corpus [13].

We considered various synchronization schemes, picking a particular k-bit scheme for each k going from 1 to 5. When k is 5 or more, it becomes possible to obtain reliable synchronization. Naturally, we chose a 5-bit scheme that provides reliable synchronization. For smaller values of k, we (rather arbitrarily) chose simplifications of the 5-bit scheme. Figure 3 describes each of the schemes we picked. The underscores denote the original bits that come from the transformed bytes.

Figure 4 presents the measurements we obtain on the benchmark files. The column titles identifies the results for BWT, PPM*C, antidictionaries, CSE without synchronization bits, and CSE when a 1- to 5-bit synchronization scheme is used, respectively. In each row, the best results are indicated in bold. Note that the measurements for BWT, PPM*C, and antidictionaries on the files `paper3`, ..., `paper6` do not appear in the original papers [4], [5]. We mention these benchmark files nevertheless since it is interesting to see the effect of synchronization on the performance of CSE itself.

We observe that, except for a few files, the more numerous the synchronization bits are, the better CSE performs. Also, the insertion of any number of synchronization bits tends to help, even when the obtained synchronization is not reliable. We do not observe any dramatic improvement when reaching the column labeled **S-5**, where reliable synchronization is used. These results suggest that CSE is able to benefit from the use of most synchronization schemes and that it does not seem to have difficulty to learn the patterns of the synchronization bits. In the case of the binary files `geo`, `obj1`, and `obj2`, we observe that synchronization improves the competitiveness of CSE, significantly reducing the gap between CSE and the leaders. The file `pic` contains binary data too but it does not benefit from synchronization. In fact, `pic` is a black-and-white image whose information is already organized at the bit level. There can be no benefit by making the byte boundaries explicit and the inserted synchronization bits only turn into an overhead for CSE.

File	BWT	PPM	Anti	CSE	S-1	S-2	S-3	S-4	S-5
bib	2.07	1.91	2.56	1.98	1.95	1.92	1.92	1.91	**1.90**
book1	2.49	2.40	3.08	2.39	2.38	**2.37**	2.39	2.42	2.43
book2	2.13	**2.02**	2.81	2.07	2.06	2.06	2.06	2.05	2.04
geo	**4.45**	4.83	6.22	5.35	5.21	4.98	4.81	4.70	4.63
news	2.59	**2.42**	3.42	2.52	2.49	2.46	2.45	2.51	2.55
obj1	**3.98**	4.00	4.87	4.46	4.53	4.43	4.32	4.24	4.17
obj2	2.64	**2.43**	3.61	2.71	2.69	2.59	2.53	2.49	2.47
paper1	2.55	**2.37**	3.17	2.54	2.51	2.48	2.47	2.46	2.44
paper2	2.51	**2.36**	3.14	2.41	2.39	2.38	2.38	2.37	**2.36**
paper3	—	—	—	2.73	2.70	2.69	2.68	2.67	**2.65**
paper4	—	—	—	3.20	3.16	3.13	3.13	3.10	**3.07**
paper5	—	—	—	3.33	3.29	3.27	3.24	3.22	**3.19**
paper6	—	—	—	2.65	2.61	2.58	2.56	2.55	**2.52**
pic	0.83	0.85	1.09	**0.77**	0.84	0.83	0.83	0.84	0.83
progc	2.58	**2.40**	3.18	2.60	2.58	2.54	2.52	2.50	2.48
progl	1.80	1.67	2.24	1.71	1.70	1.69	1.68	1.67	**1.66**
progp	1.79	**1.62**	2.27	1.78	1.76	1.73	1.71	1.70	1.68
trans	1.57	**1.45**	1.94	1.60	1.58	1.53	1.52	1.50	1.48

Fig. 4. Experimental results (in bits per character).

V. Conclusion

This paper considered the hypothesized weakness of CSE on binary data due to phase unawareness by CSE. The first question was about whether CSE really incurs a penalty due to phase unawareness when it deals with binary data. The second question was whether inserting synchronization bits inside of the data could improve CSE's performance. The answer to the first question was about half of what we expected, in the sense that it was even more positive than we thought. Indeed, CSE does incur a penalty due to the unawareness of the bits phase. We can conclude this by measuring the extent by which it is possible to improve compression by doing nothing more than providing CSE with clues about the phase. What we expected less was that CSE incurs a penalty on almost all kinds of data; it is just that the severity tends to be higher on binary data. We must concede that the experimental results cannot be seen as a *formal proof* that the hypothesis is true. Rather, it is more accurate to say that they provide strong evidence in favor of the hypothesis. This is because: the insertion of the synchronization bits cannot reduce the entropy of the original data; the inserted synchronization bits cannot hope to do more than act as phase markers; and so it would be hard to justify a claim that synchronization bits compensate for a CSE weakness other than phase unawareness. The answer to the second question is definitely positive. Inserting synchronization bits during a preprocessing step does help CSE, even though the insertion first causes the expansion of the data. We observed as a general tendency that the more numerous the inserted synchronization bits are, the more improved the compression performance is. The use of a reliable synchronization scheme is not mandatory in obtaining interesting improvements.

There are many other experiments that ought to be conducted on the phase problem of CSE. First, we only tested synchronization schemes that, at best, guaranteed synchronization after 13 bits, which is the length of a transformed byte (8 bits plus 5 synchronization bits). Since compression tends to improve with the number of inserted synchronization

bits, it would be interesting to measure the effect of using an even higher number of synchronization bits and guaranteeing synchronization on even shorter substrings.

Second, the use of more sophisticated synchronization schemes should be considered. This leads to a trade-off: obtaining better synchronization using fewer bits is an advantage but it is countered by the risk that CSE might have more difficulty to learn the synchronization bit patterns and the compression performance might suffer from it.

Third, we should consider making direct modifications to CSE to have it directly take the phase into account. For example, CSE could use *colored* bits, where one of 8 colors would be given to each bit according to its position inside of a byte. Learning the coloring scheme (e.g. the fact that a red bit would always be followed by an orange bit) should incur very little compression overhead to CSE. Indeed, once the substrings of length 2 would have been described, there would be no need to take the colors into account anymore. This modification to CSE should be relatively simple to implement.

Fourth, another direct modification to CSE would consist in describing the bit planes progressively: first, describing the bit string s_1 which is the collection of the most significant bit of every byte; then, *knowing* s_1, describing the bit string s_2 which is the collection of the two most significant bits of every byte; then, *knowing* s_2, describing the bit string s_3 which is the collection of the three most significant bits; and so on. This modification is likely to be much more complex.

Finally, there are other investigations to make on CSE that are not necessarily related to the synchronization issue. These include the prediction methods used by CSE (the best yet being an *ad hoc* learning process), a proof of the universality of CSE, and the use of the fact that there is a Hamiltonian cycle that runs through the set of l-bit substrings, for each l such that $1 \leq l \leq N$. A more complete description of this future work appears in the original paper [1].

ACKNOWLEDGMENT

We wish to thank the anonymous reviewers who contributed to improve this paper by their helpful comments. This research was funded by the Natural Science and Engineering Research Council of Canada.

REFERENCES

[1] D. Dubé and V. Beaudoin, "Lossless data compression via substring enumeration," in *Proceedings of the Data Compression Conference*, Snowbird, Utah, USA, March 2010, pp. 229–238.

[2] H. Yokoo, "The compact substring tree has linear size," Personal communications, November 2009.

[3] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.

[4] J. G. Cleary and W. J. Teahan, "Unbounded length contexts for PPM," *The Computer Journal*, vol. 40, no. 2/3, pp. 67–75, 1997.

[5] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi, "Data compression using antidictionaries," in *IEEE Special Issue on Lossless Data Compression*, 2000, pp. 1756–1768.

[6] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–342, 1977.

[7] ——, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, September 1978.

[8] M. Burrows and D. Wheeler, "A block sorting lossless data compression algorithm," Digital Equipment Corporation, Tech. Rep. 124, 1994.

[9] V. Bruyère, "A completion algorithm for codes with bounded synchronization delay," in *Proceedings of the International Colloquium on Automata, Languages and Programming*, Bologna, Italy, July 1997, pp. 87–97.

[10] L. V. Do and I. Litovsky, "On a family of codes with bounded deciphering delay," in *Proceedings of the International Conference on Developments in Language Theory*, Kyoto, Japan, September 2002, pp. 369–380.

[11] S. W. Golomb and B. Gordon, "Codes with bounded synchronization delay," *Information and Control*, vol. 8, pp. 355—-372, August 1965.

[12] R. A. Scholtz, "Codes with synchronization capability," *IEEE Transactions on Information Theory*, vol. 12, pp. 135–142, April 1966.

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