Improving Compression via Substring Enumeration by Explicit Phase Awareness

Mathieu Béliveau Danny Dubé
Université Laval, Canada
Email: Mathieu.Beliveau.2@ulaval.ca Danny.Dube@ift.ulaval.ca

Abstract

Compression by Substring Enumeration (CSE) is a recent and promising lossless compression scheme. The first experiments on CSE showed that it yields compression ratios that favorably compare to other lossless compression techniques. However, the experiments also showed that it tends to incur a performance loss on non-textual, byte-oriented sources and it was conjectured that CSE's *phase unawareness* was responsible for this loss of performance. Subsequent work confirmed the conjecture by obtaining improved compression ratios when synchronization codes get inserted in the data source, indirectly solving the phase-unawareness problem. This indirect solution does not give an absolute measure of the loss incurred by the phase unawareness problem. This paper presents a modified CSE algorithm that is made explicitly phase aware. It compares the synchronization-code approach to the explicitly phase-aware approach and shows that, in the end, the approach based on synchronization codes is almost as good as the phase-aware approach.

1 Introduction

Compression by Substring Enumeration (CSE) is a recent and promising lossless compression scheme [4]. The first experiments showed that CSE yields compression ratios that favorably compare to other lossless compression techniques. However, the experiments also showed that it tends to incur a performance loss on non-textual, byte-oriented sources. Dubé and Beaudoin conjectured that this loss of performance was caused by CSE's *phase unawareness*. Being bit-oriented, CSE is unable to take any block structure of the data into account, e.g. when the data is byte-oriented. As such, CSE is unaware of the position of the bits in their respective blocks, i.e. of the phase of the bits. Subsequent work confirmed the conjecture by obtaining better compression ratios when synchronization codes get inserted in the data source [2, 3, 9]. The approach in that work consists in preprocessing the data to insert synchronization patterns, which cause CSE to distinguish substrings that are located at different phases. It is the synchronization code that indirectly makes CSE aware of the phase. As such, the approach has the advantage of being very simple, implementation-wise, as CSE's algorithm does not have to be modified. But it is only an indirect solution to the phase-unawareness problem and one of its drawbacks is that the insertion of synchronization codes inflates the source data. Due to this effect, the actual performance of a genuinely phase-aware version of CSE remains unknown. This paper presents how CSE has been modified to be made explicitly phase aware. It compares the performance of the modified CSE to that of the original CSE with synchronization codes and shows the unexpected efficiency of the synchronization codes.

Length	Substrings							
0	8×ϵ							
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	v 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'.

The paper is structured as follows. Section 2 briefly reviews the CSE technique. Section 3 describes the problem of phase unawareness. Section 4 reviews prior work based on synchronization codes. Section 5 present our new phase-aware CSE variant. Sections 6 and 7 present and comment about the experimental comparison between the prior work and this contribution. Section 8 mentions future work.

2 Background on CSE

2.1 Preliminary Notations

For the remainder of this paper, we adopt the following conventions: $\mathbb{N}$ is the set of natural numbers; ϵ is the empty string; $|\cdot|$ designates the length of a string; a and b denote individual bits; u, v, w are strings in $\{0,1\}^*$. The data to be compressed is a binary string d of N bits. We use D to denote the circular representation of d. Informally, the circular representation of d can be thought as the infinite concatenation of d with itself.

We state that a substring w of D occurs at position p, denoted $w \in_p D$ iff: $\exists u, v \in \{0,1\}^*$, $\exists i \in \mathbb{N}$, $uwv = d^i$, and $|u| = p < N$. Note that, as D is the circular representation of d, there can be some substring w of D such that $|w| \geq N$. A substring w occurs in D, denoted by $w \in D$, iff $\exists p$ such that $w \in_p D$. Accordingly, we define the occurrence count of a substring w, C_w, to be $|\{p \in \mathbb{N} | w \in_p D\}|$. Note that by the restriction on p, we avoid counting the infinite repetitions implied by the circularity of D. Another consequence of this definition is that if $w \notin D$, then $C_w = 0$. Furthermore, in the case of $w = \epsilon$, we have $C_\epsilon = N$. Figure 1 lists the occurrence count for each substring of length at most N from $d =$ 01000001.

2.2 Overview of the Compression by Substring Enumeration Scheme

In a somewhat counterintuitive way from a compression standpoint, CSE proceeds to describe the original data string d by essentially encoding the occurrence count C_w for each lexicographically ordered substring w of D. This encoding is done in a structured manner such that, upon decoding the counts, the decompressor will be able to progressively reconstruct the original data string. Almost every step of the enumeration consists in encoding some C_{awb}, where w is called the *core* of awb and, conversely, awb is one of the (four possible) *extensions* of w.

Even if substrings of D can be of arbitrary length, in order to fully describe the original data, CSE needs only enumerate the substrings of at most N bits. Clearly, from the set of N-bit substrings, there is exactly one[1] that will match the original

[1] Assuming d is not repetitive.

data thus obviating the need to consider any longer substrings. In the following, we will cover the essential details of this description process.

From the given definition of the occurrences of some substring w of D, there is a set of relations that will lay the foundation for CSE's compression process and clarify why the mere enumeration of D's lexicographically ordered substrings can lead to a compressed representation of d. The first and foremost of these relations is that, by the circularity of D, we have:

$$C_{0w} + C_{1w} \;=\; C_w \;=\; C_{w0} + C_{w1}.$$

Given the above equality and one of the four extensions awb of w, we easily derive the three others. For instance, for a fixed C_{0w0}, we observe that:

$$\begin{array}{lcll} C_{0w1} & = & C_{0w} - C_{0w0} & \\ C_{1w0} & = & C_{w0} - C_{0w0} & \\ C_{1w1} & = & C_{w1} - C_{0w1} & = \; C_{w1} - C_{0w} + C_{0w0}. \end{array}$$

From these equalities and the fact that $C_{awb} \geq 0$, for all $a, b \in \{0, 1\}$, we obtain lower and upper bounds on C_{0w0}:

$$\max(0, C_{0w} - C_{w1}) \;\leq\; C_{0w0} \;\leq\; \min(C_{w0}, C_{0w}).$$

Assuming $|w| = n-1$, we can see from this last result that given the occurrence counts of $0w$, $1w$, $w0$, and $w1$, namely the left and right extensions of w, each of length n, we have enough information to fix the lower and upper bounds on the occurrence count for $0w0$ of length $n+1$. Also note that given C_{0w0}, C_{0w}, C_{1w}, C_{w0}, and C_{w1}, from our first set of equations, we can further deduce the occurrence count for each of the other extensions of w of length $n+1$: $0w1$, $1w0$, and $1w1$.

From this bounded prediction on C_{0w0}, we can then use any entropy encoder to lower the number of bits needed to encode each occurrence count. It is by inductively establishing those bounds on C_{0w0} from $|w| = 0$ to $|w| = N - 2$ that CSE mainly achieves compression. Furthermore, it is not all the C_{0w0} that explicitly need to be sent. For instance, if $C_{0w} = 0$ or $C_{w0} = 0$, then it can safely be assumed that $C_{0w0} = 0$. More generally, each time $\min(C_{0w}, C_{1w}, C_{w1}, C_{w0}) = 0$, there is no ambiguity on the value of C_{0w0} and thus, there is no need to explicitly send its value.

In order to describe and reconstruct the original string d only from the occurrence count of D's lexicographically ordered substrings, a special data structure, the *substring tree*, has to be maintained by both the compressor and the decompressor. The simplest form of a substring tree is what has been called the *infinite substring tree* (IST). As shown in Figure 2, each path from the root of the IST to a given node n_w forms a substring w of D; the occurrence count being n_w's label. For convenience, we identify the address of a node by the path/substring that leads to it. Referring to Figure 2 a), n_{001}, the node of address 001 would be the one and only node in solid grey. If CSE were to send each occurrence count of each substring of D in lexicographic order, it would construct the IST level by level, each time predicting C_{0w0} from the occurrence counts of the preceding level. There is however a high amount of redundancy in the IST. For instance, if, starting from the root, there is a path corresponding to the substring aw then by construction, there is a path corresponding to w. Moreover, let T_{aw} be the sub-tree rooted at n_{aw} and let T_w be the sub-tree

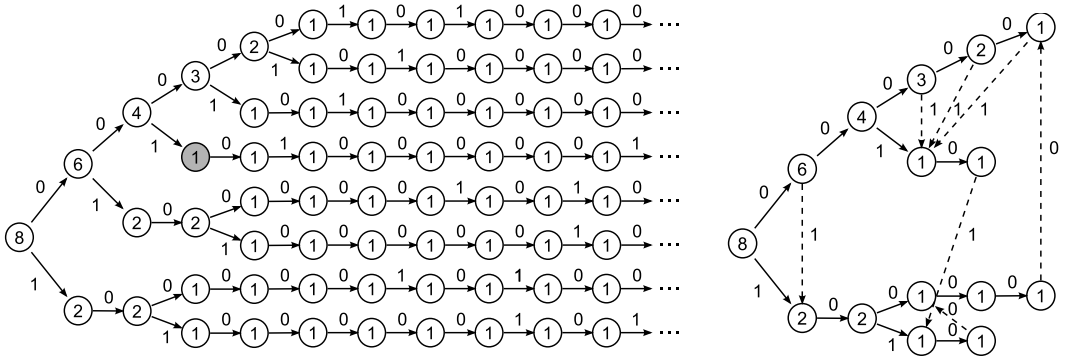

Figure 2: Substring trees for '01000001': a) the IST; b) the CST.

rooted at n_w; if $C_w = C_{aw}$, we can then assume that each and every occurrence of w is preceded by a and hence T_w and T_{aw} are isomorphic: their expansion can only lead to the same substrings. In Figure 2, we can see two such isomorphic sub-trees beginning at nodes n_{00101} and n_{0101}. It follows that if, when constructing the IST, we were to prune T_{aw} and create a shortcut from the parent of its root node to T_w, then the traversal of the resulting tree will need to emit considerably fewer occurrence counts to properly describe d. It is the result of pruning deeper isomorphic sub-trees and linking their parent node to their respective counterpart that has been called the Compacted Substring Tree (CST). In Figure 2 b), we can see how the sub-tree starting from n_{00101} has been pruned and replaced by a shortcut (a dashed line) to n_{0101}.

After having directly transmitted the occurrence counts for ϵ (the number of bits in d) and 0, CSE's algorithm initiates a breadth-first construction of the CST. Starting with ϵ as its initial core w along with the counts of its left and right extensions (0ϵ, 1ϵ, $\epsilon 0$, and $\epsilon 1$), for each step of the way, CSE extends w by estimating the bounds on C_{0w0}. If none of the occurrence counts for the left and right extensions of w is equal to zero and if there is no possible shortcut to be made to an equivalent node in the CST, CSE creates a new node and emits its bounded occurrence count using an arithmetic encoder. After reaching level N, the final step is then to send out the rank of the original string among all its lexicographically ordered rotations so that upon the reconstruction of CST, the decompressor would be able to trace the path leading to the original string.

3 The Phase Problem

As previously mentioned, experimental measurements on the performance of CSE have shown a significant loss on the compression ratio for binary (non-textual), byte-oriented data [4]. It was conjectured by the original authors that this loss of performance was the result of CSE's inherent phase unawareness. In the following, we explain what is really meant by phase unawareness and how it can affect CSE's performance on binary sources.

The notion of phase appears when the source data is organized into *blocks* of identical size. Let k be the number of bits in each block (given that, in this paper, we consider the binary alphabet only). For example, the source data may be organized into bytes and we would have $k = 8$ in that case. The *phase* of a particular occurrence of a bit in D is the offset the bit relative to the beginning of the block it is located in.

For example, in byte-oriented data, the phase of a bit is one of 0, 1, ..., 7. The *phase* of a particular occurrence of a substring in D is the phase of its leading bit.[2]

We say that CSE is *unaware of the phase* in the sense that, when it considers (all occurrences of) some substring w and its number C_w of occurrences, it does not take the phase into account. As such, when CSE encodes a number C_{awb} of occurrences, it does so while mixing the occurrences of w that are located at any phase whatsoever. Mixing the occurrences of all the phases together incurs a loss of performance by CSE. Indeed, occurrences of w at a certain phase might not have the same neighboring bits, statistically speaking, as the occurrences of w at some other phase. For example, in byte-oriented data, given some occurrences of 110 at phase 1 (which are preceded by phase-0 bits and followed by phase-4 bits) and some occurrences at phase 7 (which are preceded by phase-6 bits and followed by phase-2 bits of the *following* bytes), it seems reasonable to expect a higher correlation between phase-0 and phase-4 bits of the same bytes than between phase-6 and phase-2 bits of neighboring bytes.

Regarding CSE's performance, since CSE is essentially working at the bit level, it might not be immediately clear why there should be any difference between byte-oriented text files and byte-oriented binary data. The reason for this is rather simple: on most ASCII-encoded files, nearly all bytes have their most significant bit set to 0. When considering long enough substrings, this nearly systematic presence of a zero on the first phase of the eight-bit blocks provides a discriminating factor by which substrings taken at different phases can be distinguished by CSE. For instance, given the substring 00010001 and knowing that there cannot be a bit 1 at phase 0, among the eight possible phases of that substring, there are exactly two phase arrangements that will never occur, namely: $0_5 0_6 0_7 1_0 0_1 0_2 0_3 1_4$ and $0_1 0_2 0_3 1_4 0_5 0_6 0_7 1_0$. While the most significant bit in ASCII-encoded data is sufficient to induce some sense of phase awareness in CSE, it certainly is not reliable enough to ensure that CSE will never mix together two occurrences of same-contents substrings that are located at different phases. However, it is this idea of bits acting as markers to restrict the phase at which a given substring can appear that has lead to the scheme employed in prior work to induce a sense of phase awareness in CSE.

4 Inducing Phase Awareness Using Synchronization Codes

In order to validate the conjecture that CSE incurs a performance loss due to phase unawareness, Dubé *et al* tested an indirect solution based on synchronization codes [2, 3, 9]. The solution is said to be *indirect* because CSE's algorithm is left unchanged—and, so, technically remains phase unaware—but the insertion of synchronization codes causes CSE to cease mixing substrings of different phases together. A clear advantage of this approach is that, by not changing CSE's algorithm, it is very simple and easy to implement. The insertion of the synchronization code is performed in a preprocessing step, just before compression by CSE. On the other hand, a clear disadvantage of this approach is that the source data gets inflated by the insertion and, if the compression performance of CSE is not improved enough by the synchronization codes, then the net result might be a loss in compression performance. However, experiments showed that the insertion of synchronization codes does provide a clear improvement of the compression performance, on average. The rest of the section

[2] We may also assign a phase to a particular occurrence of ϵ in D: it has the phase of the bit that immediately follows the occurrence of ϵ.

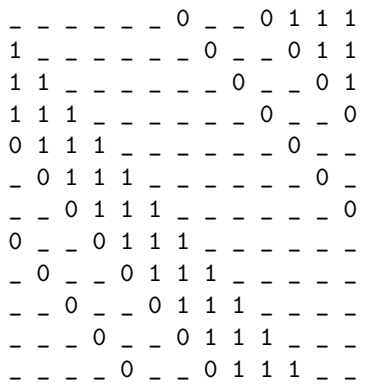

Figure 3: Demonstration of the reliability of the example pattern; $k = 8$ and $n = 5$.

gives an overview of the approach.

In this approach, a synchronization code transforms the source data on a per-block basis using a fixed pattern. Such a pattern is a sequence of jokers and control bits. A pattern contains k jokers, where k is the size of the blocks of the source data, and a certain number n of control bits. A joker is denoted by '_' and a control bit is simply a bit. For example, the pattern '_ _ _ _ _ _ 0 _ _ 0 1 1 1 1' may be used to insert synchronization codes in data organized in bytes ($k = 8$). A data block is transformed by merely copying its k bits in the pattern at the k positions marked by jokers, while preserving the order. The transformation described by a (k, n)-pattern causes an inflation of the data by a factor of $\frac{k+n}{k}$ and the transformed data in organized in blocks of $k + n$ bits. The approach based on synchronization codes uses only *reliable* ones. Reliable synchronization codes provide *sure* information about the phase, not just *probabilistic* information. Figure 3 presents a proof of the reliability of the example pattern: any two distinct rows have conflicting control bits. Thus, when we observe a substring of 13 consecutive bits (where $13 = 8 + 5$) taken at some arbitrary position in the transformed data, the phase of the substring can be identified with certainty.

When compressing transformed data, CSE benefits from clues about the phase while, at the same time, suffers from the inflation of the data caused by the transformation. CSE has to compress both the complete sequence of bits that originates from the source data and the control bits that are interspersed among them. However, Dubé *et al* selected a kind of synchronization codes, namely those based on fixed patterns of jokers and control bits, that are especially easy for CSE to *learn* and encode efficiently.

Referring to Figure 3, it is obvious that, after the insertion of the synchronization codes, any substring of length greater or equal to 13 can only be located on a unique phase: there cannot be two identical substrings having 13 or more bits which would begin at different phases. Thus, for $|w| \geq 13$, when predicting and encoding C_{0w0} based on C_{0w}, C_{1w}, C_{w0}, C_{w1}, the phase of all the occurrences of w is unique, even if unknown to CSE *per se*, so no encoding on substrings of mixed phases happens. Moreover, if the bit to the left of w happens to be a synchronization bit, say bit 1, due to the pattern used by the synchronization code, it follows that each and every occurrence of w will have 1 to its left, resulting in $C_{1w} = C_w$, and $C_{0w} = 0$. As

previously shown, a count of zero in either one of w's left or right extension leads to the immediate inference of C_{0w0} and hence the opportunity not to send that information. This effect is the result of CSE *having learned the synchronization code*. Predicting and encoding the control bits can be performed by CSE at *no cost* for cores w such that $|w| \geq 13$. Learning the synchronization code incurs a cost only for cores w such that $|w| < 13$.

5 Inducing Phase Awareness Explicitly

Although CSE can effectively learn synchronization codes, the cost of inserting those codes in-between the original data can hardly be null. Here, we present a modification to CSE's algorithm that allows it to be phase aware without incurring any inflation on the original data and allowing phase synchronization for each and every substring, regardless of their length.

First, let us slightly refine our definition of occurrence for some substring w of D to take the phase into account. Given blocks of length k, we say that some substring w occurs at phase q and position p, noted $w \in_p^q D$, if $\exists i \in \mathbb{N}$, $\exists u, v \in \{\mathtt{0}, \mathtt{1}\}^*$ such that $uwv = d^i$, $|u| \bmod k = q$, and $|u| = p < N$. That is to simply say that some substring w occurs at phase q if and only if it begins at offset q from the beginning of a block and begins at a position that is less than the length of the original string d. For short, we note w_q to refer to some substring beginning at phase q regardless of its position. Correspondingly, we define the number of occurrences for some substring w at phase q, noted C_w^q, as $|\{p \in \mathbb{N} | w \in_p^q D\}|$.

In this context, the original equations that allowed us to predict the bounds for C_{0w0}, now noted C_{0w0}^q, are also slightly modified. But, first, we introduce an abbreviated notation: $q \oplus \delta$ means $q + \delta \bmod k$. Now, we have:

$$C_{0w}^q + C_{1w}^q \;=\; C_w^{q\oplus 1} \;=\; C_{w0}^{q\oplus 1} + C_{w1}^{q\oplus 1}.$$

From which we can derive:

$$\begin{array}{rcl} C_{0w1}^q &=& C_{0w}^q - C_{0w0}^q \\ C_{1w0}^q &=& C_{w0}^{q\oplus 1} - C_{0w0}^q \\ C_{1w1}^q &=& C_{w1}^{q\oplus 1} - C_{0w1}^q \;=\; C_{w1}^{q\oplus 1} - C_{0w}^q + C_{0w0}^q. \end{array}$$

Hence, our fundamental inequality to establish the bounds on C_{0w0}^q becomes:

$$\max(0, C_{0w}^q - C_{w1}^{q\oplus 1}) \;\leq\; C_{0w0}^q \;\leq\; \min(C_{w0}^{q\oplus 1}, C_{0w}^q).$$

Having defined the bounds on C_{0w0}^q, we now need to define exactly which of those occurrence count will have to be sent. This means that the CST on which we previously relied for this purpose will have to be modified.

As before, the CST is built by enumerating the occurrences of each lexicographically ordered substring of D with the difference that all those occurrences, as per the new definition of occurrence, have to be counted under their respective phase. It follows that for a given phase q, the empty substring ϵ will now have as many occurrences as there are blocks of length k in d. This means that our CST will now have k root nodes, each corresponding to some ϵ_q substring, and such that $C_\epsilon^q = \frac{N}{k}$. For each of those roots, still as before, CSE will directly emit the occurrence count

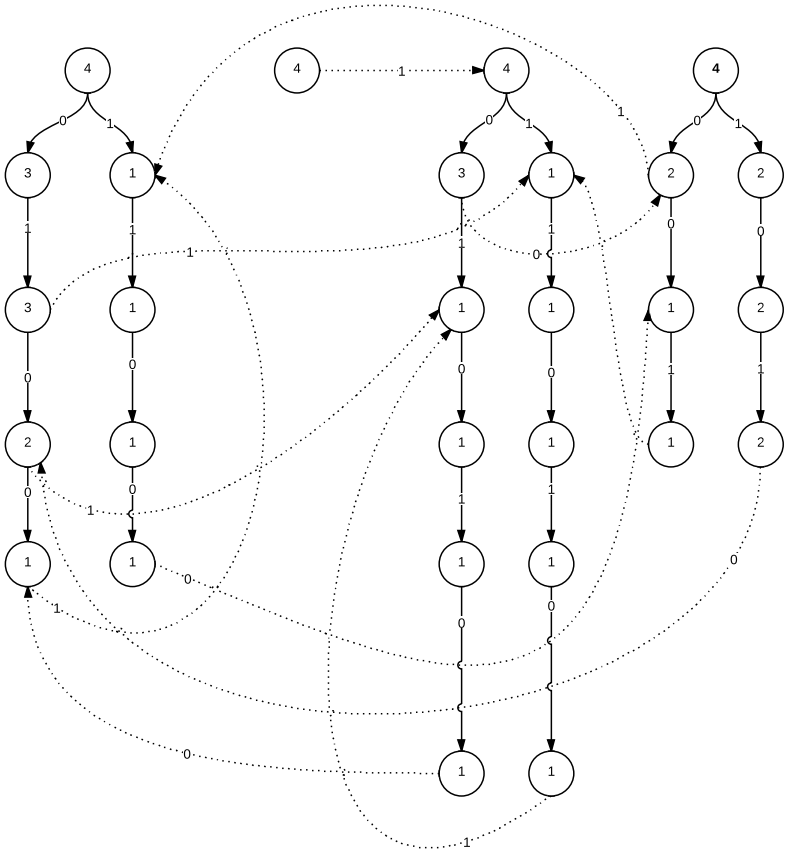

Figure 4: The 4-phase CST for the binary encoding of 'uL': '0111 0101 0100 1100'.

for ϵ_q and $\mathtt{0}_q$. It will then proceed to predict the bounds of $C^q_{\mathtt{0}w\mathtt{0}}$, beginning with $w = \epsilon$ and create the resulting CST nodes up to $|w| = N - 2$.

Recall that a CST is the compacted expression of an IST, namely that while the IST explicitly develops the isomorphic sub-trees that results from two substrings having a common suffix and the same occurrence count, the CST will prune the sub-tree associated with of the longest substring and create a shortcut to the root of the sub-tree at the end of the shortest substring. When taking the phase into account, this means that while building a core's extension for a given phase, we also have to look on the previous level of the next phase if we can find a common suffix with the same occurrence count. The sub-tree rooted at n^q_{aw} is isomorphic to the one rooted at $n^{q\oplus 1}_w$ if $C^q_{aw} = C^{q\oplus 1}_w$. A simple illustration for this is given in Figure 4: because $C^1_{\mathtt{1}\epsilon} = C^2_\epsilon$, instead of expanding $n^1_{\mathtt{1}\epsilon}$'s sub-tree, we directly link to n^2_ϵ. Having a procedure to predict occurrence counts and accordingly build a CST while always taking the phase into account, CSE can effectively be said to be explicitly phase aware.

6 Experimental Results

Here we present a comparison of the compression efficiency on the files of the Calgary Corpus [10]; comparing `gzip` [6] set at maximal compression, the compression ratios for PPM*C and the Burrows-Wheeler transform (BWT) reported in [1], the original implementation of CSE, CSE with optimal Synchronization Codes [9] (noted CSE SC)

File	Gzip	BWT	PPM	CSE	CSE SC	CSE EPA
bib	2.51	2.07	1.91	1.98	1.88	**1.87**
book1	3.25	2.49	2.40	2.27	2.33	**2.24**
book2	2.70	2.13	2.02	1.98	**1.93**	**1.93**
geo	5.34	**4.45**	4.83	5.35	4.57	4.56
news	3.06	2.59	**2.42**	2.52	**2.42**	**2.42**
obj1	**3.84**	3.98	4.00	4.46	3.99	3.95
obj2	2.63	2.64	**2.43**	2.71	2.44	2.44
paper1	2.79	2.55	**2.37**	2.54	2.41	2.39
paper2	2.89	2.51	2.36	2.41	2.34	**2.33**

File	Gzip	BWT	PPM	CSE	CSE SC	CSE EPA
paper3	3.11	—	—	2.73	2.63	**2.61**
paper4	3.33	—	—	3.20	3.01	**2.96**
paper5	3.34	—	—	3.33	3.10	**3.05**
paper6	2.77	—	—	2.65	2.49	**2.47**
pic	0.82	0.83	0.85	**0.77**	0.81	0.81
progc	2.68	2.58	**2.40**	2.60	2.44	2.42
progl	1.80	1.80	1.67	1.71	1.64	**1.63**
progp	1.81	1.79	**1.62**	1.78	1.66	1.64
trans	1.61	1.57	**1.45**	1.60	1.47	**1.45**

Figure 5: Experimental results.

and CSE with Explicit Phase Awareness (noted CSE EPA). The measurements (in bits per character) are presented in Figure 5. Note that CSE usually divides too large a file into several chunks to be individually compressed. In the following benchmark, each of CSE's variants used chunks of 3 MB, hence being able to process each of the Calgary's corpus files in one pass, with or without synchronization codes.

7 Discussion

As can be seen from the above results, in the few instances where CSE EPA does not in fact provide a better compression ratio, the gap between the best listed algorithms and CSE EPA is very small.

A somewhat peculiar result however, is that of the compression ratios obtained from the `pic` file. It is the only instance where the original implementation of CSE actually outperforms both of its phase-aware variants. The `pic` file, being a binary image, is indeed the only file from the Calgary Corpus where the data is not byte-oriented. From this context, grouping substring occurrences based on their phase is a false premise and it might very well be (as it is here the case) that this artificial phase restriction on the bounds of the occurrences of some substring will result in worst compression ratios than if they were accounted for regardless of their phase.

Another interesting and rather unexpected result is the small and sometimes nearly null margin between CSE with synchronization codes and its explicitly phase-aware counterpart. As expected, the explicitly phase-aware version of CSE always leads to better compression ratios. Yet, the smallness of the gap between the two versions clearly shows how effective CSE is in learning the synchronization codes and exploiting the implicit phase awareness provided by them.

8 Future Work

One important detail that has been overlooked in the general presentation of CSE is the probability distribution of C_{0w0} between its lower and upper bounds. At the moment, CSE's algorithm uses a statistical learner that has proved to be consistently better than simply assuming a flat probability distribution, thus leading to better compression ratios.

However, the effectiveness of this statistical learner has not been established and some knowledge on the evolution of the statistical distribution of C_{0w0}, especially in upper levels of the CST, could very well enhance CSE's general performance. Most

of the research on CSE is currently done in that direction. In theory, CSE has been shown to be universal for Markovian sources and, more generally, for stationary and ergodic sources [5, 11]. The demonstrations assume that another statistical distribution is used by CSE, namely the hypergeometric distribution and a related one based on necklaces. Iwata *et al* have proposed to use slightly stronger bounds on C_{0w0} [7]. They also showed that the version of CSE described in Dubé and Yokoo's proof of universality suffers from little redundancy [8].

Acknowledgements

This work is supported by the Natural Science and Engineering Research Council of Canada.

References

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

[2] Danny Dubé. Using synchronization bits to boost compression by substring enumeration. In *Proceedings of the International Symposium on Information Theory and its Applications*, pages 82–87, Taichung, Taiwan, October 2010.

[3] Danny Dubé. On the use of stronger synchronization to boost compression by substring enumeration. In *Proceedings of the Data Compression Conference*, page 454, Snowbird, Utah, USA, March 2011.

[4] Danny Dubé and Vincent Beaudoin. Lossless data compression via substring enumeration. In *Proceedings of the Data Compression Conference*, pages 229–238, Snowbird, Utah, USA, March 2010.

[5] Danny Dubé and Hidetoshi Yokoo. The universality and linearity of compression by substring enumeration. In *Proceedings of the International Symposium on Information Theory*, pages 1519–1523, Saint-Petersburg, Russia, July 2011.

[6] Jean-Loup Gailly and Mark Adler. The GZIP compressor. `http://www.gzip.org`.

[7] Ken-ichi Iwata, Mitsuharu Arimura, and Yuki Shima. An improvement in lossless data compression via substring enumeration. In *Proceedings of the IEEE/ACIS International Conference on Computer and Information Science*, pages 219–223, Sanya, Hainan Island, China, May 2011.

[8] Ken-ichi Iwata, Mitsuharu Arimura, and Yuki Shima. On the maximum redundancy of CSE for i.i.d. sources. In *Proceedings of the International Symposium on Information Theory and Applications*, pages 489–492, Honolulu, Hawaii, USA, October 2012.

[9] Dany Vohl, Claude-Guy Quimper, and Danny Dubé. Finding synchronization codes to boost compression by substring enumeration. In *Proceedings of the International Workshop on Constraint Modelling and Reformulation*, Quebec City, Quebec, Canada, October 2012.

[10] Ian Witten, Timothy Bell, and John Cleary. The Calgary corpus, 1987. `ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus`.

[11] Hidetoshi Yokoo. Asymptotic optimal lossless compression via the CSE technique. In *Proceedings of the International Conference on Data Compression, Communications and Processing*, pages 11–18, Palinuro, Italy, June 2011.