Optimal encoding of non-stationary sources

John H. Reif [a], James A. Storer [b]

[a] *Duke University, USA*
[b] *Computer Science Department, Brandeis University, Waltham, MA 02254, USA*

Abstract

The usual assumption for proofs of the optimality of lossless encoding is a stationary ergodic source. Dynamic sources with non-stationary probability distributions occur in many practical situations where the data source is formed from a composition of distinct sources, for example, a document with multiple authors, a multimedia document, or the composition of distinct packets sent over a communication channel. There is a vast literature of adaptive methods used to tailor the compression to dynamic sources. However, little is known about optimal or near optimal methods for lossless compression of strings generated by sources that are not stationary ergodic. Here, we do not assume the source is stationary. Instead, we assume that the source produces an infinite sequence of concatenated finite strings $s_1 \ldots s_n$, where:

(i) Each finite string s_i is generated by a sampling of a (possibly distinct) stationary ergodic source S_i, and

(ii) the length of each of the s_i is lower bounded by a function $L(n)$ such that $L(n)/\log(n)$ grows unboundedly with the length n of all the text within $s_1 \ldots s_i$.

Thus each input string is a sequence of substrings generated by possibly distinct and unknown stationary ergodic sources. The optimal expected length of a compressed coding of a finite prefix $s_1 \ldots s_k$ is

$$\sum_{i=1}^{k} n_i H_i,$$

where n_i is the length of s_i and H_i is the entropy of S_i. We give a window-based LZ77-type method for compression that we prove gives an encoding with asymptotically optimal expected length. We give another LZ77-type method for compression where the expected time for encoding and decoding is nearly linear (approaching arbitrarily close to linear $O(n)$ for large n). We also prove that this later method gives an encoding with

***Preprint of paper appearing* in Special Issue of Information Sciences, Volume 135, pp. 87-105 (2001).**

asymptotically optimal expected length. In addition, give a dictionary-based LZ78-type method for compression, which takes linear time with small constant factors. This final algorithm also gives an encoding with asymptotically optimal expected length, assuming the S_i are stationary ergodic sources that satisfy certain mixing conditions and $L(n) \geqslant n^{\varepsilon}$ for some $\varepsilon > 0$.

1. Introduction

1.1. Conventional LZ compression methods

The family of *LZ compression methods* of Lempel and Ziv is a class of lossless string compression methods that use variable-rate coding. For simplicity, we assume all input sources to be strings over the same finite alphabet and all compressed strings to be binary. The LZ compression methods are *universal* coding methods in the sense that the compression algorithm executes without initial knowledge of the source distribution. A *parsing* of a string is a division of the string into substrings called *phrases*. A *distinct* parsing is a parsing such that no two phrases are identical. An LZ compression method makes a distinct parsing of the input into phrases, and each of these phrases is encoded using an index into a *dictionary* that contains previously parsed phrases followed by a character. We refer to such indices as *pointers*. Pointers represent the *matched* portion of the phrase, which is a string that matches the incoming input. The character portion of a phrase is the character of the input that follows the *matched* portion.

1.2. Compression methods using a finite window

In Ziv and Lampel [46], the adaptive dictionary consists of a *window* of the preceding n characters, and the matched portion of the current phrase is the longest possible substring within that window.

1.3. Compression methods using an adaptive dictionary

In Ziv and Lampel [47], an adaptive dictionary stores previously parsed phrases, which indicate a substring anywhere in the previous characters. The matched portion of the current phrase is a (longest possible) phrase stored in the dictionary. After the phrase consisting of the index of that matched portion together with the next input character is sent to the decoder, that phrase is added to the dictionary. The decoder receives a new phrase consisting of an index and a character, outputs that phrase, and adds that phrase to the

dictionary. Thus, the encoder and decoder are maintaining in "lock-step" identical copies of a dynamically growing dictionary of strings.

Both LZ77 and LZ78 are provably optimal in the information theoretic sense for ergodic sources. At each step, both LZ77 and LZ78 send a match (a position-length pair of numbers for LZ77 or a single dictionary index for LZ78) followed by an uncompressed character. The inclusion of a raw character provides a simple mechanism to guarantee that progress is always made (since each phrase must have length at least one) and that each phrase of the parsing is distinct (since each phrase is one character longer than a longest possible match). Sending a raw character with each pointer is useful for proving optimality, but not necessary (see Storer and Reif [30]).

In practice, to avoid the overhead of one uncompressed character for each matched string, codes can be reserved for the characters of the input alphabet to guarantee that all phrases have length at least one. This has no change on the matching process of LZ77, and for LZ78 one can still add the current match together with the next input character as a new dictionary entry (but this next input character is not part of the current phrase), and the decoder can effectively work one step behind the encoder to deduce this next character (e.g., the "LZW" method of the UNIX compress utility works this way). We will denote methods that reflect LZ77 operation as window-based or LZ77-type and methods that reflect LZ78 operation as dictionary-based or LZ78-type methods. It is also typical in practice to limit LZ77-type methods to a "window" of finite length and limit LZ78-type methods to a dictionary of finite size (by either freezing it when it is full or by incorporating a deletion method). Another practical variation is to allow non-greedy parsing; that is, use a shorter than possible match at a given step to allow for a longer match at a later step. See the books of Storer [29] and Bell et al. [3] for a presentation of other variations and practical implementations. For simplicity, we limit our attention to "pure" LZ77 and LZ78. However, all of the techniques we present can be generalized to incorporate most practical implementations of LZ77 and LZ78, but if the variation in question gives up optimality, then so will our methods.

1.4. Efficient implementation techniques for LZ compression

McCreight [21] presented a linear time and space algorithm for the construction of a suffix tree, which effectively represents all substrings of a given string in linear space and allows an LZ77 algorithm to find a longest match in constant time per character read, which yields a linear time implementation. Rodeh et al. [24] presented a linear time implementation for LZ77 string matching for a bounded-size sliding window, by employing three overlapping suffix tries. Fiala and Greene [1989] [14] presented a modification to McCreight's approach for a sliding window, where vertices are continuously deleted

in (amortized) constant time per character read, where refinements for strict real-time implementation are made in Ukkonen [39]; see the Ph.D. thesis of Larsson [17] for further references. Others, such as Brent [8] and the patents of [40] and Whiting [1991] [41] employed hashing to find matches in a sliding window. For LZ78 methods, a simple trie can be used to store the dictionary, and matching can be done in constant time per character read by traversing a root to leaf path in the trie as characters of the match are read (e.g., Welch [42], Storer [28,29], Miller and Wegman [22]). Royals et al. [25] and Storer and Reif [31] describe a massively parallel implementation of a systolic LZ compression scheme also using pair-wise recursive parsing. Storer and Reif [30] describe the adaptation of LZ techniques to insure error resiliency.

1.5. Stationary source models for lossless compression

Bell et al. [4] discuss many source models for lossless text compression. A number of models for string sources are now listed in increasing generality:

(i) *Symmetric Bernoulli sources*, where each symbol is independently randomly generated with equal likelihood;

(ii) *Asymmetric Bernoulli sources*, where each symbol is independently randomly generated with a different probability;

(iii) *Finite Markov sources*, where the string is generated by a finite Markov process;

(iv) *Finite Memory sources*, where the string is generated by an ergodic probabilistic finite state process, where the probability of a given character is dependent only on the current state;

(v) *α-mixing ergodic stationary sources*, as defined in Szpankowski [37], where the string is generated by any ergodic stationary process, that satisfy α-mixing conditions, where for any strings A of length n and B of length $n+d$

$$(1-\alpha(d))\,\mathrm{Prob}(A)\,\mathrm{Prob}(B) \leqslant \mathrm{Prob}(AB) \leqslant (1+\alpha(d)) \leqslant \mathrm{Prob}(A)\,\mathrm{Prob}(B),$$

where $\alpha(d)$ is a quantity such that $\alpha(d) \to 0$ as $d \to \infty$.

(vi) *Ergodic stationary sources*, as defined in Cover and Thomas [11].

Note that in each of these models, the source string to the data compression algorithm is generated by an unknown ergodic stationary process.

1.6. Previous analysis of lossless compression algorithms

Gallager [15] and also Cover and Thomas [11] give excellent introductions to basic notions (such as of entropy) in information theory and data compression. Let the *compression factor* of the compression of a length n string be the average encoded length per source symbol, that is, the compression length divided by the length n of the string. Let the *optimality ratio*

$\gamma(n)$ of the compression of a length n string be the ratio of the compression factor divided by the entropy H of the source. The compression is *asymptotic optimal* in the information theoretic sense if the optimality ratio $\gamma(n) \to 1$ as the text string length $n \to \infty$. Lempel and Ziv [18] (see also Seery and Ziv [26,27]) gave an analysis of LZ compression, which was first viewed as a method for measuring the complexity of a string. Ziv and Lempel [46] proved their LZ77 algorithm, which used a window of text as the dictionary, is as good as any other of a general class of adaptive methods for lossless data compression. Ziv and Lempel [47] proved their dictionary-based LZ78 algorithm gives asymptotically optimal compression for strings generated by any ergodic stationary process. This was a significant milestone result in the theory of lossless data compression. Wyner and Ziv [45] discuss asymptotic properties of the entropy of a stationary ergodic source. Cover and Thomas [11] later gave a particularly elegant proof of the asymptotic optimality of LZ78 compression for ergodic stationary processes. There has been considerable further research on the analysis of LZ methods. Plotnik et al. [23] gave a more refined asymptotic analysis of the LZ algorithm for finite-state sources. Louchard and Szpankowski [20] give a probabilistic analysis of the lengths of paths in the suffix tree of the LZ78 parsing scheme, for stationary ergodic sources. Szpankowski [35,36] analyzes the asymptotic properties of the depth of LZ suffix trees for memoryless sources, and Szpankowski [37] extends this analysis to α-mixing ergodic stationary sources. Also, Abrahamson [1] gave an adaptive dependency source model for related data compression methods.

1.7. SSES dynamic sources

Let a *dynamic source* be a source that is not stationary. Here we assume the source is a particular class of dynamic sources, where we are motivated by practical situations where the data source is formed from a composition of distinct sources (for example, a document with multiple authors, a multimedia document, or the composition of distinct packets sent over a communication channel).

We assume that the source produces an infinite sequence of concatenated finite strings $s_1, s_2, \ldots$ where:

(i) Each finite string s_i (a *stationary component*) is generated by a sampling of a (possibly distinct) stationary ergodic source S_i (a *stationary component source*).

(ii) The length of each of the s_i is lower bounded by a function $L(n)$ such that $L(n)/\log(n)$ grows unboundedly with the length n of all the previous text within $s_1 \ldots s_i$.

Thus, each source string (input to the data compression algorithm) is formed from a sequence of substrings generated by possibly distinct and unknown

stationary ergodic sources. We will call such a source a *SSES dynamic source*, where *SSES* denotes a sequence of stationary ergodic sources.

Let n_i be the length of s_i and let H_i be the entropy of stationary ergodic source S_i. As usual, we assume that H_i is constant independent of n_i. The optimal expected length of a compressed coding of a finite prefix $s_1 \ldots s_k$ is:

$$\sum_{i=1}^{k} n_i H_i.$$

Expected Length Lemma. *The expected length of each text phrase within s_i using LZ77 limits to* $\log(n_i)/H_i$ *as* $n \to \infty$.

Proof. The average encoded length per source symbol is H_i, and each pointer has length $\log(n_i) + O(n_i)$, so the expected length of each text phrase in s_i using LZ77 has limit $\log(n_i)/H_i$ as $n \to \infty$. □

Recall that LZ algorithms have been shown to have asymptotic optimal compression for stationary ergodic sources, so their optimality ratio approaches 1 as the text string length approaches infinity. Hence, to provide asymptotically optimal compression to an SSES dynamic source, it suffices to provide asymptotically optimal compression to each component source. However, these component sources S_i are initially unknown to the compression algorithm; also the position and even the length of the stings s_i they generate is assumed to be initially unknown. This lack of knowledge is the key difficulty of optimally compressing SSES sources. We also define a class α-MSSES of dynamic sources which are defined to be SSES dynamic sources where each stationary component source S_i is an α-mixing ergodic stationary source (as defined above).

1.8. Our results and organization of this paper

Section 1 gave basic definitions, described previous compression methods for stationary sources, and stated our results. Section 2 presents a relatively simple algorithm using a window-based LZ77-type method which has asymptotic optimal compression, but is inefficient with respect to time. Section 3 then presents an alternative algorithm also using a LZ77-type method and gives methods for efficient execution of that algorithm with near linear expected time; approaching arbitrarily close to $O(n)$ for large n. Section 4 gives a linear time bounded dictionary-based LZ78-type algorithm which has asymptotically optimal expected compression for α-MSSES dynamic sources, when $L(n) \geqslant n^{\varepsilon}$ for some $\varepsilon > 0$. Section 5 shows how to find maximal matches in text windows in expected linear time using a total of $O(n \log\log(n))$ space.

Section 6 concludes the paper with a discussion of open problems and of further applications.

2. Asymptotically optimal compression for SESS dynamic sources

We first give a very simple proof of asymptotic optimal compression for SSES dynamic sources, using a window-based LZ77-type method with variable-length prefix codes.

2.1. Encoding pointers by variable-length prefix codes

In practice it is often the case that for simplicity all indices encoding pointers are represented using a fixed-length binary code. However, since the windows may grow in length quickly, it is more advantageous for our approach to use a variable-length prefix code for positions within the window, where each pointer is represented by an encoding with a variable number of bits, and its prefix can be distinguished from any other encoded number. A straightforward approach is to encode an integer $i \geqslant 1$ with $2\lfloor \log(i) \rfloor + 1$ bits by prefixing the encoding with a unary representation of the length (the log of i) followed by the binary representation of i, but this doubles the number of bits. By applying this idea recursively, the overhead goes to zero in the limit. The idea is to use a "cascading lengths" code where the number is preceded by its length, the length by its length, and so on until a constant number of bits is reached; the total length of the code is $\lfloor \log(i) \rfloor + \lfloor \log\log(i) \rfloor + \cdots + 1$ bits; see Levenshtein [19], Ellias [12], and Even and Rodeh [13]. This technique was also used by Rodeh et al. [24] in their sliding window implementation, and described (and given the name cascading lengths) in the book of Storer [29]. Extensions of this idea to near optimal bounds are given in Bentley and Yao [7]. Given that we are using cascading lengths codes, it is convenient to represent a match in a LZ77 system as a *displacement-length* pair, where the *displacement* specifies how many characters back the match starts and the length specifies the number of characters in the match. Note that this does not affect the optimality of the LZ77 method since the use of the cascading lengths code adds no asymptotic overhead, and then the conversion to a displacement-length pointer representation can only shorten the representation of phrases.

2.2. Window-based methods with variable-length prefix codes

We now adapt a window-based LZ77-type method with variable-length prefix coding of pointers to an SSES dynamic source. So each source string input to the data compression algorithm is formed from a sequence of finite strings $s_1, s_2, \ldots$, where each stationary component s_i is a finite string generated

by a stationary ergodic source S_i. Recall that n_i denotes the length of s_i and H_i denotes the entropy of stationary ergodic source S_i. Also by definition of an SSES dynamic source, n_i is lower bounded by a function $L(n)$ such that $L(n)/\log(n)$ grows unboundedly with n. The optimal expected length of a compressed coding of a finite prefix $s_1 \ldots s_k$ of length n is:

$$\sum_{i=1}^{k} n_i H_i.$$

To compress the sequence of strings $s_1 \ldots s_k$, we treat it as a single string and run the standard LZ77 algorithm (with displacement-length pointer encoded with cascading lengths) with the following change: at each step, instead of using the longest possible match, use the match that has the smallest compression ratio (number of bits used to represent the match divided by the number of characters in the match).

Now consider a particular string s_i. Let us compare the compression we get on s_i (after already having compressed $s_1 \ldots s_{k-1}$) and call this the *joint compression* of s_i to what we would have got by compressing s_i individually, call this the *individual compression* of s_i. Since displacement-length pointers with cascading lengths are being used, all pointers available for individual compression are also available for joint compression at exactly the same cost. So the naive conclusion is that since at each step joint compression chooses pointers with minimum compression ratio, it must compress at least as well as individual compression. However, there are two problems:

(a) *Optimality*. When the joint compression uses a match that is smaller than what is used by individual compression, the subsequent portion of the source string is pre-conditioned by this choice and standard proofs of optimality may not apply.

(b) *Complexity*. The time may no longer be linear because at any given step, we may search down very deep in the suffix trie only to discover that longer matches do not have better compression ratios (because they are too far back).

Reducing the time complexity will be addressed in the next section. Although it is possible that more powerful techniques could show this approach to be optimal, we can work around the potential problem of optimality by processing data in blocks. On each block, we perform an optimal parse (e.g., using dynamic programming from right to left in the block along the lines of Storer and Szymanski [1978, 1982] [32,33] – see also the book of Storer [29]). The optimal parse of the joint compression uses the same or fewer bits as the individual compression of the block. The only added cost is when a pointer of the individual compression overlaps a block boundary and is "charged" to two blocks in this analysis; but this gives a net increased cost of at most one pointer per block, for the cost of the joint compression over the individual compres-

sion. It suffices to choose the block size in any way that makes the number of pointers processed per block a function that grows unboundedly with n. But this is implied by the Expected Length Lemma and our definition of an SSES dynamic source, where n_i is lower bounded by a function $L(n)$ such that $L(n)/\log(n)$ grows unboundedly with n.

3. Fast asymptotically optimal compression for SESS sources

The previous section introduced a construction to adapt window-based LZ compression to a dynamic source and achieve compression that is optimal in the information theoretic sense. However, the complexity was problematic because the algorithm employed a single suffix trie for the entire past history. When compressing a particular source string s_i, at any step it was possible to traverse a very deep path in the suffix trie, only to end up finding that a relatively short match gave the best compression ratio for that step. Here, we describe another window-based LZ77-type algorithm using a more "constructive" approach to searching for matches, and again using variable-length prefix codes. We review universal hashing, use hashing techniques for the match search, give a space efficient method match search, and complete the description of this algorithm. We continue to assume an SSES dynamic source where each source string input to the data compression algorithm is formed from a sequence of finite strings $s_1, s_2, \ldots$ (where each stationary component s_i is a finite string generated by a stationary ergodic source S_i with entropy H_i), and the length n_i of each s_i is lower bounded by $L(n)$, where $L(n)/\log(n)$ grows unboundedly with n.

To develop a fast algorithm, we again adapt a window-based LZ77-type method with variable-length prefix coding of pointers. Let $R(n)$ be a non-constant monotonic increasing positive function of n where $R(n) = \mathrm{O}(n)$ and $R(n) \leqslant n$. For example, $R(n)$ might be defined to be the n^{ε} for some $0 < \varepsilon < 1$.

To motivate our algorithm, first suppose, we somehow determine a substring w of length $R(L(n)) + m$ within a stationary component s_i. We place a window of initial length $R(L(n))$ starting at the first character of w, and then parse by LZ77 algorithm a sequence of successive phrases (where just after each phrase, the window is enlarged to the right by the length of the phrase, as usual in LZ77-type algorithms) of total length m. Then, if the number of phrases grows unboundedly with n, this sequence of successive LZ77 phrases can be shown to have asymptotically optimal compression, for in this case the optimality ratio $\gamma(R(L(n)))$ approaches 1 as $n \to \infty$ (this follows since, the LZ77 optimality ratio has limit $\gamma(n)$ approaches 1 as $n \to \infty$ and $R(L(n))$ grows unboundedly with n).

To see that the number of phrases grows unboundedly with n, a sufficient condition is that $m/\log(n)$ grows unboundedly with n. By The Expected Length Lemma, the expected length of each text phrase within s_i using LZ77

limits to $\log(n_i)/H_i$ as $n_i \to \omega$. Hence, if $m/\log(n)$ grows unboundedly with n, then the number of phrases is

$$\frac{m}{\log(n_i)/H_i} = \Omega\left(\frac{m}{\log(n)}\right) \geq \Omega\left(\frac{L(n)}{\log(n)}\right),$$

which grows unboundedly with n.

Thus to provide asymptotically optimal compression to an SSES dynamic source, it suffices to provide asymptotically optimal compression to small windows within each component source. However, these component sources S_i are initially unknown to the compression algorithm; even the length of the stings s_i they generate is assumed to be initially unknown. Thus, we will try a number of window lengths and use that window size that provides the best overall compression for a sequence of subsequent LZ77 phrases.

Let $L'(n)$ be a function where both $L(n)/L'(n)$ and $L'(n)/\log(n)$ grow unboundedly with n. Let $R^{(j)}(n)$ be the result of applying $R(n)$ to itself j times. Let $u(n)$ be the j such that $R^{(j)}(n) \leqslant 2$; that is, the number of times $R(n)$ can be applied to itself until it is $\leqslant 2$. For example, if $R(n) = n^{\varepsilon}$ for some $0 < \varepsilon < 1$, then:

$$u(n) = \log_{1/\varepsilon} \log(n).$$

We further require that $R(n)$ grow small sufficiently quickly so that $u(n)$ grows sufficiently slowly so that $L(n)/(u(n)\log(n))$ grows unboundedly with n (e.g., let $u(n) = L'(n)/\log(n)$).

We now will describe a text compression algorithm that as usual in LZ77, processes the text incrementally from left to right. Let the current position within the source text be p. We construct a sequence of $u(n)+1$ windows, where the right end of each initially is just preceding the current source text position p. For each $j = 0, 1, \ldots, u(n)$ the window W_j is initially of length $R^{(j)}(n)$. For each such j, we determine the next maximal phrase (longest match with a substring of the window) that would be constructed by a LZ77-type method using this window W_j, and let m_j be the length of this phrase. Let $m_{\max}$ be the length of the longest such phrase, and m be the larger of $m_{\max}$ and $L'(n)$. Let NEXT(m) be the sequence of source text of length m from the current text position p. Next for each $j = 0, 1, \ldots, u(n)$, we parse the text strictly within NEXT(m) exactly as LZ77 would using window W_j. That is, we determine a sequence of maximal phrases strictly within NEXT(m) using the rightward-growing window W_j, as usual in LZ77-type methods. Each phrase will be represented by a pointer defining the length of the phrase and displacement of its match with in the rightward-growing window W_j. These pointers representing a phrase within NEXT(m) using window W_j will be encoded by a variable-length prefix codings, and will together be called the jth *candidate encoded pointer sequence*. Let E_j be the length of this encoding. Then, we choose for our incremental compression output that candidate encoded pointer sequence where E_j is the smallest among all of these $0 \leqslant j \leqslant \lceil \log(n) \rceil$.

Furthermore, if the candidate encoded pointer sequence does not end at the last character of NEXT(m), we also output the phrase just following these previous phrases, using rightward-growing window W_j. This *following* phrase may overlap some final text of NEXT(m) and some subsequent text past the characters of NEXT(m). Then we repeat this compression process from the new resulting source text position at the right end of the most recent phrase. Note that the choice of a given window, say W_j, does not precondition the subsequent source distribution, since this choice of the window W_j is made by considering only parses of phrases strictly within NEXT(m), and the subsequent output of the following phrase insures that the resulting source text position is outside of NEXT(m).

Now we show this compression method is asymptotically optimal for SSES dynamic sources.

By definition, NEXT(m) has length $m \geqslant L'(n)$. Also by definition, $L'(n)/\log(n)$ grows unboundedly with n, so $m/\log(n)$ grows unboundedly with n. By The Expected Length Lemma, the expected length of each text phrase using LZ77 limits to at most:

$$\frac{\log(n_i)}{H_i} = \mathrm{O}(\log(n_i)) \leqslant \mathrm{O}(\log(n)) \text{ as } n_i \to \infty.$$

Thus the number of phrases parsed within NEXT(m) limits to

$$\geqslant \frac{m}{\mathrm{O}(\log(n))} \geqslant \Omega\left(\frac{m}{\log(n)}\right),$$

which grows unboundedly with n.

Now observe that the number of phrases parsed within NEXT(m) grows unboundedly with n. Let j^* be the largest number such that the length $R^{(j^*)}(n)$ window W_{j^*} (just preceding the current source text position) is entirely within a component string s_i generated by a stationary source S_i. Note that this definition of j^* implies that

$$R^{(j^*)}(n) \geqslant R(n_i) \geqslant R(L(n)),$$

since $R(n)$ is non-decreasing with $R(n) \leqslant n$, and $n_i \geqslant L(n)$. Also, let us assume that the subsequent NEXT(m) is also entirely within s_i; note that:

- In this the case, and if the window W_{j^*} is chosen by our algorithm, then the compression of the parsed phrases within NEXT(m) using the window W_{j^*} has optimality ratio:

  $$\gamma(R^{(j^*)}(n)) \leqslant \gamma(R(L(n))).$$

  (This follows since LZ77 compression of stationary ergodic sources has optimality ratio $\gamma(n) \to 1$ as $n \to \infty$, and also $R^{(j^*)}(n) \geqslant R(n_i) \geqslant R(L(n))$ and $n_i \geqslant L(n)$.)

- When this is not the case, then the loss of compression is not significant asymptotically. By The Expected Length Lemma, each m_j expects to be at most $\log(n_i)$, so $m_{\max}$ expects to be at most $O(u(n)/\log(n))$. Hence, when m is the larger of $m_{\max}$ and $L'(n)$, it expects to be $\leqslant$ the larger of $O(u(n)\log(n))$ and $L'(n)$. By definition of $L(n)$ and $u(n)$, the ratios $L(n)/L'(n)$ and $L(n)/(u(n)\log(n))$ both grow unboundedly with n. So $n_i/m \geqslant L(n)/m$ also grows unboundedly with n, which insures the compression is still asymptotically optimal.

But $R(L(n))$ grows unboundedly with n, so $\gamma(n) \to 1$ as $n \to \infty$. Furthermore, if another window W_j other than window W_{j^*} is chosen, then the encoding of the phrases within NEXT(m) using window W_j must be at least as compact as those using W_{j^*}, since this is the criteria for choosing window W_j. This implies that the compression of the phrases within NEXT(m) is asymptotically optimal.

Note that, if the candidate encoded pointer sequence does not end at the last character of NEXT(m), we also output the next phrase using window W_j just following these previous phrases. Since this next phrase may overlap some final text of NEXT(m) and some subsequent text outside of NEXT(m), that phrase may not be optimally encoded. However, the loss of compression is not significant asymptotically, since the number of phrases parsed within NEXT(m) grows unboundedly with n.

Hence, we conclude this algorithm's compression of an SSES dynamic source is asymptotically optimal.

3.1. Universal hashing

Carter and Wegman [10] first developed universal classes of hash functions, that provide a very small number of hash conflicts by the technique of choosing a hash function randomly from a universal class of hash functions. These universal hash codes can be used for efficient text matching, as required for LZ77 compression. A universal-2 hash family F_n of Carter and Wegman [10] is defined as follows: let $\sigma \geqslant 2$ be a fixed integer equal to the number of bits needed to encode a symbol of the alphabet (i.e., the log of the alphabet size). Each element of F_n is a hash function h mapping from strings of length n to the integers $\{0 \ldots \sigma n - 1\}$. A pair of distinct keys x and y in $\{0 \ldots n-1\}$ have a *hash conflict* with respect to hash function h in F_n if $h(x) = h(y)$. Carter and Wegman [10] define and provide a universal-2 hash family F_n so that if we choose a random hash function h in F_n, then for each key x in $\{0 \ldots n-1\}$, the probability that there exists a y in $\{0 \ldots n-1\}$ such that $x \neq y$ and $h(x) = h(y)$ is $\leqslant 1/\sigma$. This implies that using a randomly chosen hash function in a universal-2 hash family, for worst case choice of strings to hash, the expected time per hash is O(1) (using say hash buckets or linked lists to resolve the expected O(1) conflicts

per key, and so that no items are lost). Thus $O(n)$ expected time is required to hash any n items.

A hash family is *associative* if each of the hash functions of the family can be associatively applied to substrings A and B to form the hash of the string AB from the hash of A and B. Carter and Wegman [10] also provide a universal-2 family of associative hash functions.

3.2. Finding maximal matches in text windows

Brent [8] gave a method for LZ77 string matching using incremental hashing, that hashes each match prefix using an incremental associative hash function until the longest match is found. However, any such method using hashing (even if a hash function randomly chosen from a universal-2 family of hash functions is used) needs to verify that we have not found a false match (due to hash collisions), and this may take linear time in the length of the match.

Here, we assume a hash function is randomly chosen from a universal-2 family of associative hash functions, with parameter $\sigma = \log(n)$. Given any text string A, we perform a straight forward linear incremental search for the longest prefix of A matching with a substring of the window W. However, we verify that we have found a correct partial match only of every $\log(n)$ of the matches, and we also verify the final match (which is to be the maximal parse). Each hash has a collision with probability $1/\sigma = 1/\log(n)$, and so the likelihood, that there is no hash collision after $\log(n)$ hashes of partial matches, is $(1 - 1/\log(n))^{\log(n)}$ which limits to $1/e$ (where e denotes the natural logarithm base – see a standard calculus text for the derivation of this limit such as Thomas [38]) as $n \to \infty$. If there is a hash collision on a partial match, then again in expected time linear in the length of the match, we can go back and determine the correct match (if any).

Let λ be the length of the resulting longest match, which is the next phrase of the LZ77 parse. By The Expected Length Lemma, λ expects to be $\leqslant O(\log(n))$. Note that we do not need to assume that W_j and NEXT(m) are within a single stationary ergodic source, since otherwise, if say the window has the wrong length, then the maximal match will expect to be at least as small. Hence in the search for the longest prefix of A, the expected number of verified partial matches is $O(1)$. Thus the total expected time to find a prefix match of maximal length λ within the window W is $O(\lambda)$.

The hash tables are assumed to be empty initially, so take no time to initialize (this is a standard assumption used in many data structure algorithms). Thus, we take expected linear time but use a total of total $O(n\log(n))$ space. Section 5 shows how to find maximal matches in text windows in expected linear time using a total of $O(n\log\log(n))$ space.

3.3. Efficient execution of our compression algorithm

Recall that the compression method of this section considers $u(n)+1$ windows of preceding source text, and requires the hashing and determination of the maximal match of subsequent text with substring of the windows W_j. We do this using the algorithm just presented in $O(\lambda)$ expected time per match of length λ, or O(1) expected time per input character. The total expected time to do the maximal match parsing in all the $u(n)+1$ windows during the algorithm is $O(nu(n))$. Note that we can let $u(n) \to O(1)$ as $n \to 0$, using a sequence of with slower and slower (monotonically) increasing (but still non-constant) functions $R(n)$. Thus, the total expected time for compressing the length n input source string approaches (but does not reach) $O(n)$ for $u(n) \to O(1)$. We also can decompress the encoded string within these time bounds using nearly identical techniques.

4. Linear time optimal compression for α-MSSES dynamic sources

We now briefly describe a dictionary-based LZ78-type method for compression, which gives an encoding with asymptotically optimal expected length, in the case where the S_i are α-mixing ergodic stationary sources. This algorithm always takes linear time with small constant factors.

LZ78-type methods typically use tries (root-to-leaf paths represent strings in an arbitrary set of strings with the prefix property). The adaptive dictionary is represented by a trie which contains phrases previously parsed since the dictionary was last reinitialized, and the depth of the trie bounds the longest phrase previously parsed (again, since the dictionary was last reinitialized). Our LZ78-type method for compression will use a dictionary which is periodically reinitialized (made empty) by deleting all nontrivial entries. Szpankowski [37], showed that, for any α-mixing ergodic stationary source, with probability

$$1 - \frac{c\log(n)}{n^{\varepsilon}},$$

the depth of the suffix tree is at most

$$(1+\varepsilon)\frac{1}{h}\log(n)$$

for any $\varepsilon > 0$ and some constants c and h. Also, Szpankowski [37] showed that, with probability

$$\geqslant 1 - \frac{c'\log(n)}{n^{\varepsilon}},$$

the length of a random LZ77 phrase is at least

$$-\varepsilon + \frac{1}{h}\log(n)$$

for any $\varepsilon > 0$ and some constant c' and the same constant h. A similar result can be shown for phrases resulting from the dictionary-based LZ78 algorithm.

The probabilistic lower bounds on phrase length will be used to determine when to reinitialize the dictionary. That is, we will reinitialize the dictionary when:

(i) The number of source string characters input since the last reinitialization of the dictionary is at least $L(n)^{\varepsilon}$, for some $\varepsilon > 0$, and

(ii) there has been a further sequence of at least $L(n)^{\varepsilon}$ most recent source string characters where each phrase is of length at most

$$-\varepsilon + \frac{1}{h}\log(L(n))^{\varepsilon} = \varepsilon\left(\frac{\log(L(n)}{h} - 1\right)$$

for some appropriate constant h.

Note that Case (i) insures that reinitializing of the dictionary is not considered until a sufficiently long interval, and since each stationary segment s_i is assumed to be of least length $L(n)$, a length $L(n)^{\varepsilon}$ interval will insure that at least the compression of this current stationary segment is near optimal, from which the asymptotic optimality of the algorithm follows. Since this compression algorithm is identical to the usual LZ78 implementations with periodic dictionary reinitialization (except in the conditions for reinitialization), it follows (by known implementations of LZ78) that the total time cost for the compression of a length n string is $O(n)$. We also can decompress the encoded string within these linear time bounds using nearly identical techniques.

5. Finding maximal matches in small space

Note that the compression algorithm of Section 3 finds maximal matches in text windows in expected linear time using a total of total $O(n\log(n))$ space. Here, we show that we can finding maximal matches in text windows in expected linear time using a total of $O(n\log\log(n))$ space. We again assume a hash function is randomly chosen from a universal-2 family of associative hash functions, using parameter $\sigma = \log\log(n)$. The hash tables are again assumed to be empty initially, so take no time to initialize (again we note this assumption is typical for data structure algorithms). Given a source text string of length n over an SSES dynamic source, we define a hierarchical *binary recursive subdivision* of this string, which consists of $\lceil\log(n)\rceil$ levels. For each j in $\{0\ldots\lceil\log(n)\rceil\}$ we partition the string into a set P_j of segments of maximal length $\leqslant 2^j$. As we are incrementally input the source string, we incrementally

hash each of those segments of each of the P_j that have been input. Since the total number of segments in

$$P = \bigcup_{j=0}^{\lceil \log(n) \rceil} P_j$$

is $\leqslant 2n$, the total expected time to hash all the segments of P is $O(n)$. (Note that the hashing of the segments of (shifted) binary recursive subdivision can easily be maintained over a shifting window, since the binary recursive subdivision needs no modification on shifts other than hashing the new segments that need to be added.)

Thus, a text string of any length λ is composed of at most $2\log(n)$ substrings in P, and so can be hashed in $O(\log(\lambda))$ time, assuming (as we have) that those substrings in P have already been hashed. Furthermore, given any text string A, we can perform a straight forward binary search for the longest prefix of A matching with a substring of the window W. Let λ be the length of the resulting longest match. Assuming we have already hashed all segments of a binary partitioning of a text window W of length n, given any text string A, we easily can execute this $O(\log \lambda)$ stage binary search guided by the binary recursive subdivision, and using only $O(\log \lambda)$ further associative hash function applications. The sum of the lengths of the partial matches during this binary search is at most twice the length λ of the maximal match. By The Expected Length Lemma, the expected length λ of the maximal match is $O(\log(n))$. Note that we do not need to assume that W_j and NEXT(m) are within a single stationary ergodic source; for otherwise the maximal match will expect to be at least as small), and the expected number of stages of the binary search is $\log(O(\log n)$.

Again, we need to verify that we have not found a false match (due to hash collisions). During the binary search, we verify partial matches only every $\log\log(n)$ partial matches, and we again also verify the final match (which is to be the maximal parse). Each hash has a collision with probability $\leqslant 1/\sigma = 1/\log\log(n)$, and so the likelihood, that there is no hash collision after $\log\log(n)$ hashes of partial matches, is

$$\left(1 - \frac{1}{\log\log(n)}\right)^{\log\log(n)},$$

which again limits to $1/e$ (where e denotes the natural logarithm base) as $n \to \infty$. Again, if there is a hash collision on a partial match, then in expected time linear in the length of the match, we can go back and determine the correct match (if any).

Thus the total expected time to find a prefix match of maximal length λ within the window W is $O(\lambda)$ (not counting the time to initially hash the elements of the binary recursive subdivision of W, which is $O(n)$).

6. Conclusion and open problems

We have given a number of asymptotically efficient algorithms that address, at least from the theoretical point of view, optimal lossless compression of dynamic sources. These techniques can also be applied to a number of applications not related to compression. Weinberger et al. [1992] [43] discuss the use of LZ schemes for discrimination between sequences. Vitter and Krishnan [34] describe the use of lossless data compression techniques to do optimal prefetching for certain sources. Both these results can be extended to dynamic source strings using our techniques. The most important open problem is to determine the largest possible class of dynamic sources where we can provide lossless compression within linear or near linear time. Also, empirical testing on actual sources is needed to further refine our dynamic source models and compression algorithms.

7. For further reading

Refs. [2,5,6,9,16,44].

References

[1] D.M. Abrahamson, An adaptive dependency source model for data compression, Communications of the ACM 32 (1) (1989) 77–83.
[2] T. Bell, A. Moffatt, A note on the DMC data compression scheme, The Computer Journal 32 (1) (1989).
[3] T. Bell, J.G. Cleary, I.H. Witten, Text Compression, Prentice-Hall, Englewood Cliffs, NJ, 1990.
[4] T. Bell, I.H. Witten, G.J. Cleary, Modeling for text compression, ACM Computing Surveys 24 (1) (1989) 557–591.
[5] P.E. Bender, J.K. Wolf, New asymptotic bounds and improvements on the Lempel-Ziv data compression algorithm, IEEE Transactions on Information Theory 37 (3) (1991) 721–729.
[6] J.L. Bentley, D.D. Sleator, R.E. Tarjan, V.K. Lei, A locally adaptive data compression scheme, Communications of the ACM 29 (4) (1986) 320–330.
[7] J.L. Bentley, A. Yao, An almost optimal algorithm for unbounded searching, Information Processing Letters 5 (3) (1976) 82–87.
[8] R.P. Brent, A linear time algorithm for data compression, Australian Computer Journal 19 (2) (1987) 64–68.
[9] Bookstein, S.T. Klein, T. Taita, I.K.R, Rao, M.D. Patil, Can random fluctuation be exploited in data compression, Symposium on the IEEE Symposium On Data Compression, 1993, pp. 70–78.
[10] J.L. Carter, M.N. Wegman, Universal classes of hash functions, Journal of Computer and System Sciences 18 (1979) 143–154.
[11] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley, New York, 1991.

[12] P. Elias, Universal codeword sets and representations of the integers, IEEE Transactions on Information Theory 24 (5) (1975) 194–203.
[13] S. Even, M. Rodeh, Economical encoding of commas between strings, Communications of the ACM 21 (4) (1978) 315–317.
[14] E.R. Fiala, D, H. Greene, Data Compression with Fixed Windows, Zerox Palo Alto Res. Cent, 1998.
[15] R.G. Gallager, Information Theory and Reliable Communication, Wiley, New York, 1968.
[16] T. Hagerup, C. Rub, A guided tour of Chernoff bounds, Information Processing Letters 33 (1989) 305–308.
[17] N.J. Larsson, Structures of String Matching and Data Compression, Ph.D. Thesis, Dept. of Computer Science, Lund University, Sweden, 1999.
[18] A. Lempel, J. Ziv, On the complexity of finite sequences, IEEE Transactions on Information Theory 22 (1) (1976) 75–81.
[19] V.E. Levenshtein, On the redundancy and delay of separable codes for the natural numbers, Problems in Cybernetics 20 (1968) 173–179.
[20] Louchard, Szpankowski, Generalized Lempel–Ziv parsing scheme and its preliminary analysis of the average profile, Symposium on the IEEE Symposium on Data Compression, 1995, pp. 262–271.
[21] E.M. McCreight, A space-economical suffix tree construction algorithm, Journal of the Association for Computing Machinery 23 (2) (1976) 262–272.
[22] V.S. Miller, M.N. Wegman, Variations on a theme by Lempel and Ziv, Combinatorial Algorithms on Words, 1985, pp. 131–140.
[23] E. Plotnik, M.J. Weinberger, J. Ziv, Upper Bounds on the Probability of Sequences Emitted by Finite-State Sources and on the Redundancy of the Lempel-Ziv Algorithm, 1991.
[24] M. Rodeh, V. Pratt, S. Even, Linear algorithm for data compression via string matching, Journal of the Association for Computing Machinery 228 (1) (1981) 16–24.
[25] D.M. Royals, T. Markas, N. Kanapoulos, J.H. Reif, J.A. Storer, On the design and implementation of a lossless data compression and decompression chip, IEEE Journal of Solid-State Circular 28 (9) (1989) 948–953.
[26] J.B. Seery, J. Ziv, A universal data compression algorithm: description and preliminary results, Technical Memorandum 77 (1977) 1212–1216.
[27] J.B. Seery, J. Ziv, Further results on universal data compression, Technical Memorandum 78 (1978) 1212–1218.
[28] J.A. Storer, Textual substitution techniques for data compression, in: Combinatorial Algorithms on Words, 1985, pp. 111–129.
[29] J.A. Storer, Data Compression: Methods and Theory, 1988.
[30] J.A. Storer, J.H. Reif, Error resilient optimal data compression, SICOMP, 1997.
[31] J.A. Storer, J.H. Reif, A parallel architecture for high speed data compression, Journal of Parallel and Distribute Computing 13 (1991) 222–227.
[32] J.A. Storer, T. Szymanski, The macro model for data compression, Proceedings of the Tenth Annual ACM Symposium on the Theory of Computing, 1982, pp. 30–39.
[33] J. Storer, T. Szymanski, Data compression via textual substitution, Journal of the Association for Computing Machinery 29 (4) (1982) 928–951.
[34] J.S. Vitter, P. Krishnan, Optimal prefetching via data compression, Brown University CS Dept. Technical Report No. CS-91-46, 1991.
[35] W. Szpankowski, A typical behavior of typical suffix trees, IEEE Symposium on Data Compression, 1991, pp. 247–256.
[36] W. Szpankowski, (Un)expected behavior of typical suffix trees, in: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 422–431.
[37] W. Szpankowski, Asymptotic properties of data compression and suffix trees, IEEE Transactions on Information Theory 39 (5) (1993) 1647–1659.

[38] G.B. Thomas, Calculus and Analytic Geometry, Addison-Wesley, Reading, MA, 1968.
[39] E. Ukkonen, On-line construction of suffix trees, Technical Report A-1993-1, Dept. of Computer Science, University of Helsinki, 1993.
[40] J.R. Waterworth, Data Compression Systems, 1987.
[41] D.A. Whiting, G.A. George, G.E. Ivey, Data Compression Apparatus and Method, 1991.
[42] T.A. Welch, A technique for high-performance data compression, IEEE Computer 17 (6) (1984) 8–19.
[43] M.J. Weinberger, J. Ziv, A. Lempel, On the optimal asymptotic performance of the universal ordering and discrimination of individual sequences, in: Proceedings of IEEE Symposium on Data Compression, 1991, pp. 239–245.
[44] A. Wyner, J. Ziv, Fixed data base version of the Lempel-Ziv data compression algorithm, IEEE Symposium on Data Compression, 1991, pp. 203–207.
[45] A. Wyner, J. Ziv, Some asymptotic properties of the entropy of a stationary ergodic source with applications to data compression, IEEE Transactions on Information Theory 35 (6) (1989) 1250–1258.
[46] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Transactions on Information Theory 23 (3) (1977) 337–343.
[47] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory 24 (5) (1978) 530–536.