Error Resilient LZ'77 Scheme and Its Analysis*

Stefano Lonardi
Dept. Computer Science & Eng.
University of California
Riverside, CA 92521
stelo@cs.ucr.edu

Wojciech Szpankowski
Dept. of Computer Sciences
Purdue University
West Lafayette, IN 47907
spa@cs.purdue.edu

Mark Daniel Ward
Dept. of Mathematics
Purdue University
West Lafayette, IN 47907
mward@math.purdue.edu

Abstract

We propose here a joint source-channel coding algorithm capable of correcting errors in the popular Lempel-Ziv'77 scheme without practically losing any compression power. This is possible since the LZ'77 encoder does not completely remove all redundancy. The inherent additional redundancy left by the LZ'77 encoder is used succinctly by a channel coder (e.g., Reed-Solomon coder) to protect against a limited number of errors. In addition to this, the scheme proposed here is perfectly backward-compatible, that is, a file compressed with our error-resilient LZ'77 can be still decompressed by a common LZ'77 decoder. We present the algorithm and its preliminary analysis.

1 Introduction

Error-resilient adaptive lossless data compression is a particularly challenging problem because of two opposing "forces". *Source coding* tries to decorrelate as much as possible the input sequence (i.e., by removing redundant information), while *channel coding* introduces additional correlation (i.e., by adding redundant information) in order to protect against errors. The devastating effect of errors in adaptive data compression is a long-standing open problem (cf. [8]). In fact, the non-resilience of adaptive data compression has been a practical drawback of its use in many applications. Joint source-channel coding has emerged as a viable solution to this problem.

Here we deal with the best-known adaptive data compression scheme, namely that of Lempel-Ziv'77 [13]. This popular compression scheme works on-line and replaces the longest prefix of the not-yet-compressed portion of a file with a *pointer* and *length* of its copy in the already compressed file. The lack of error-resilience of LZ'77 is a well-recognized problem.

How much overhead is needed in LZ'77 to correct errors? We shall argue that practically there is no need for additional overhead in order to correct errors in LZ'77. This seemingly impossible goal is achieved in practice due to the fact that the LZ'77 encoder is unable to decorrelate completely the input sequence. Some implicit redundancy is still present in the compressed stream and can be exploited by the encoder. The additional redundancy derives from the encoding of phrases for which one has a choice among $r > 1$ possible pointers. In practice, if there are r copies of the longest prefix, we recover $\lfloor \log_2 r \rfloor$ redundant bits by choosing one of the r pointers.

We observe theoretically and experimentally (cf. [4]) that in a significant proportion of phrases, there is more than one copy of the longest prefix in the compressed file. More precisely, we shall

[1]This work was supported in part by NSF Grants CCR-0208709, DBI-0321756 and NIH grant R01 GM068959-01.

indicate that the number of copies M_n of the longest match in a sequence of length n generated by a binary memoryless source (with p being the probability of generating "0") follows the *logarithmic series distribution* (plus some fluctuations), that is, $P(M_n = j) \approx (1/h)(p^j(1-p) + (1-p)^j p)/j$ where h is the entropy rate.

Once the redundant bits of LZ'77 have been identified, one can devise a method to exploit them for channel coding. We choose $RS(255, 255-2e)$ Reed-Solomon codes, where 255 is the size of the buffer containing the data and the parity bits, and e is the maximum number of errors that the code can correct. We should point out that if e is large then we may not *always* have enough redundant bits to embed the parity bits. We are currently working on a scheme in which e is changed adaptively with the availability of redundancy bits in the stream.

To the best of our knowledge, the scheme described here is the first joint source-channel LZ'77 algorithm. In [8] Storer and Reif address the issue of *error propagation* but not error recovery (cf. see [6] for an analysis of the Storer and Reif algorithm). There are, however, joint source-channel coding algorithms for arithmetic coding and other variable length codes (see, e.g., [7]).

In addition to the description of the scheme, we provide a novel theoretical analysis. The results presented here are proved using precise tools of the analysis of algorithms (cf. [9]).

2 Augmenting LZ'77 with Error Resilience

Let T be a text of length n over a finite alphabet $\mathcal{A}$. We write $T_{[i]}$, $1 \le i \le n$, to indicate the i-th symbol in T. We use $T_{[i,j]}$ as shorthand for the substring $T_{[i]}T_{[i+1]} \dots T_{[j]}$ where $1 \le i \le j \le n$, with the convention that $T_{[i,i]} = T_{[i]}$. Substrings of the form $T_{[1,j]}$ correspond to the prefixes of T, and substrings of the form $T_{[i,n]}$ to the suffixes of T.

The LZ'77 algorithm [13] processes the data *on-line* as it is read, i.e., it parses the file sequentially *left to right* and looks into the sequence of past symbols to find a match with the longest prefix of the string starting at the current position. The longest prefix is substituted with a *pointer*, which is a triple composed of *(position, length, symbol)*. Several variations on LZ'77 have been proposed (see, e.g., [2] and references therein), but the basic principle remains the same.

Let us suppose that the first $i-1$ symbols of the string T have been already parsed in $k-1$ phrases, i.e., $T_{[1,i-1]} = y_1 y_2 \dots y_{k-1}$. To identify the k-th phrase, LZ'77 looks for the *longest* prefix of $T_{[i,n]}$ that matches a substring of $T_{[1,i-1]}$. If $T_{[j,j+l-1]}$, $j < i$ is the substring that matches the longest prefix, then the next phrase is $y_k = T_{[i,i+l-1]}$. The algorithm issues the pointer $(j, l, T_{[i+l]})$ and updates the current position i to $i+l+1$. The reason we need $T_{[i+l]}$ is to be able to advance when $l = 0$, which is common in the very beginning.

In order to recover additional bits to be used for channel coding, we slightly modify the LZ'77 encoding. The resulting algorithm, called LZS'77, allows one to embed some bits of another text M over the alphabet $\{0,1\}$. We define a position i corresponding to the beginning of a phrase to have *multiplicity* r if there exist exactly r matches for the longest prefix that starts at position i in T. The positions with multiplicity $r > 1$ are the places where we can embed some of the bits of M. Specifically, the next $\lfloor \log_2(r) \rfloor$ bits will drive the selection of one particular pointer out of the r choices. These additional bits can be used for various purposes such as authentication [1] or error correction as described next.

Suppose again that the initial portion of T, say $T_{[1,i-1]}$, has been already parsed. Let $\{(p_0, l, T_{[i+l]}), (p_1, l, T_{[i+l]}), \dots, (p_{r-1}, l, T_{[i+l]})\}$, $r \ge 1$, be the set of feasible pointers for the longest prefix of $T_{[i,n]}$, where $l > 1$, and $1 \le p_l \le i$ for all $0 \le l \le r-1$. If $r = 1$ we skip to the next phrase, and no extra bits are embedded. When $r > 1$, we use the next $d = \lfloor \log_2(r) \rfloor$ bits of M to choose one of the r pointers. Suppose that the first $t-1$ bits of M have been already embedded in previous phrases. We emit the pointer $(p_{M_{[t,t+d]}}, l, T_{[i+l]})$, we move the current position to $i+l+1$, and we increment t by d.

We now describe how the extra redundant bits can be exploited to achieve error resilience. Since we are protecting the stream of pointers, which is typically represented by a sequence of bytes, we use Reed-Solomon (RS) codes [5]. A Reed-Solomon code is specified as RS(a, b), where a is the size of the block and b is the size of the payload. The datum is a symbol drawn from an alphabet of cardinality 2^s. The encoder collects b symbols and adds $a-b$ parity symbols to make a block of length a. A Reed-Solomon decoder can correct up to e errors in a block, where $e = (a-b)/2$. One symbol error occurs if one or more of the bits of the symbol (up to s) is wrong.

Given a symbol size s, the maximum block length a for a Reed-Solomon code is $a = 2^s - 1$. For example, the maximum length of a code with 8-bit symbols ($s = 8$) is 255 bytes. The family of Reed-Solomon codes for $s = 8$ is therefore RS$(255, 255 - 2e)$. Each block contains 255 bytes, of which $255 - 2e$ are data and $2e$ are parity. Errors up to e bytes anywhere in the block can be automatically detected and corrected.

We can use the extra redundancy bits of LZS'77 to embed $2e$ extra bytes, as described in the following. The encoder, called LZRS'77, first compresses T using standard LZ'77. The data is broken into blocks of size $255 - 2e$. Then, blocks are processed in reverse order, beginning with the very last. When processing block i, the encoder computes first the Reed-Solomon parity bits for the block $i+1$ and then it embeds the extra bits in the pointers of block i using LZS'77 idea described above. The parity bits of the first block are not embedded, but saved at the beginning of the compressed file.

The decoder receives a sequence of pointers, preceded by the parity bits of the first block. It first breaks the input stream into blocks of size $255-2e$. Then it uses the parity bits to correct the first block. Once block B_1 is correct, it decompresses it using LZS'77. This not only reconstructs the initial portion of the original text, but it also recovers the bits stored in those particular choices for the pointers. These extra bits are collected, and they become the parity bits for the second block. The decoder can therefore correct possible errors in B_2. Block B_2 is then decompressed, and the parity bits for B_3 are recovered. This process continues until all blocks have been decompressed.

The reason the encoder needs to process the blocks in reverse order should now be apparent. The encoder cannot compute the RS parity bits before the pointers are finalized. We embed the RS bits for the current block in the *previous* block, because the decoder needs to know the parity bits of a block *before* it attempts to decompress it. This has the unfortunate effect of making the encoder off-line, since it requires the encoder to keep the entire set of buffers in primary memory. The problem can be alleviated by breaking up large inputs into pieces of a size that could be easily stored and processed in main memory.

3 Analysis of LZS'77

We only present preliminary analytical results leading to the size of redundant bits in the LZS'77 which is the first step to analyze fully LZRS'77. More precisely, we formulate a result that will allow us to estimate the distribution of the number of copies of the longest prefix when the text is generated by a memoryless source.

Let $T_{[1,n]}$ be the first n symbols generated by the source and let L_n be the random variable associated with the length of the longest prefix (phrase) of $T_{[n+1,\infty]}$ which has an occurrence in $T_{[1,n]}$. In other words, the random variable L_n describes the length of the phrases of LZ'77. Henceforth, we will use L instead of L_n to simplify the notation.

We associate the variable W_n to the multiplicity of the pointer at position n in the text, that is, $W_n = \sum_{i=1}^{n-L} \mathbf{1}(T_{[i,i+l-1]} = T_{[n+1,n+L]})$. Using the results by Wyner [12] it is easy to prove the following theorem.

Theorem 1 *For Markov sources,* $\mathbf{E}[W_n] = O(1)$.

In the rest of this section, the objective is to obtain a more precise characterization of the asymptotic distribution of W_n and its moments. Observe that one can define W_n in terms of the associated suffix trie S_n built from the first n suffixes of the text T. In fact, when inserting the $(n+1)$-st suffix of T into S_n the size of the subtree starting at the branching point of a new insertion is exactly W_n.

In order to study the variable W_n we first reduce the problem to a simpler one that, asymptotically, is equivalent to our problem. Instead of analyzing a random suffix tree we construct a random trie built from n *independently generated* strings generated by a memoryless source. It is known that such a trie approximates well the initial suffix trie [3] [1]. As a consequence of this, we can now concentrate our analysis to tries. Let us define a parameter M_n such that the asymptotic distribution of M_n is the same as W_n.

We first give a mathematically rigorous formulation of the problem to be analyzed. Let p be fixed with $0 < p < 1$, and write $q = 1 - p$. Define $X(j)$ to be the string $X_1^{(j)} X_2^{(j)} X_3^{(j)} \dots$, where $\{X_i^{(j)} \mid i, j \in \mathbb{N}\}$ is a collection of i.i.d. random variables on $\{0, 1\}$, with $P\{X_i^{(j)} = 0\} = p$. Let $l_j^{(n)} = \sup\{i \geq 0 \mid X_1^{(j)} \dots X_i^{(j)} = X_1^{(n+1)} \dots X_i^{(n+1)}\}$. Then define $L_n = \max_{j \leq n} l_j^{(n)}$. In other words, among the first n strings, L_n denotes the length of the longest common prefix with the $(n+1)$-st string. Finally, define $M_n = \#\{j \mid 1 \leq j \leq n,\ l_j^{(n)} = L_n\}$. So M_n counts how many of the first n strings have a common prefix of length L_n with the $(n+1)$-st string. By convention, let $M_0 = 0$. We observe that if the strings $X(1), \dots, X(n)$ are suffixes of a text T, then M_n is expected to be asymptotically equivalent to W_n defined above.

Now we present the problem from the viewpoint of tries. The alignment $C_{j_1,\dots,j_k}$ among k strings $X(j_1), \dots, X(j_k)$ is the length of the longest common prefix of the k strings. We observe that $l_j^{(n)} = C_{j,n+1}$. The k-th depth $D_{n+1}(k)$ in a trie built over $n+1$ strings is the length of the path from the root of the trie to the leaf containing the k-th string. Since $D_{n+1}(n+1) = \max\limits_{1 \leq j \leq n} C_{j,n+1} + 1$, then $L_n = D_{n+1}(n+1) - 1$. Thus, in the context of tries, $M_n = \#\{j \mid 1 \leq j \leq n,\ C_{j,n+1} + 1 = D_{n+1}(n+1)\}$. That is, M_n is the size of a subtree starting at the branching point of a new insertion.

Define the exponential generating functions

$$G(z, u) = \sum_{n \geq 0} E[u^{M_n}] \frac{z^n}{n!} \qquad \text{and} \qquad W_j(z) = \sum_{n \geq 0} E[(M_n)^{\underline{j}}] \frac{z^n}{n!}$$

for complex $u \in \mathbb{C}$ and $j \in \mathbb{N}$. If $f : \mathbb{C} \to \mathbb{C}$, then the recurrence relation

$$\begin{aligned} E[f(M_n)] &= p^n (q f(n) + p E[f(M_n)]) + q^n (p f(n) + q E[f(M_n)]) + \\ &\quad + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} \left(p E[f(M_k)] + q E[f(M_{n-k})] \right) \end{aligned}$$

[1] Indeed, if one builds a suffix tree from $n / \log n$ suffixes separated by $\log n$ symbols, then such a tree is asymptotically equivalent to an independently built trie.

holds for all $n \in \mathbb{N}$. If $f(0) = 0$, then the recurrence also holds when $n = 0$. To verify the recurrence, just consider the possible values of $X_1^{(j)}$ for $1 \le j \le n+1$. Two useful facts follow immediately from this recurrence relation. First, if $n \in \mathbb{N}$, then

$$E[u^{M_n}] = p^n(qu^n + pE[u^{M_n}]) + q^n(pu^n + qE[u^{M_n}]) + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (pE[u^{M_k}] + qE[u^{M_{n-k}}]). \quad (1)$$

Also, if $j \in \mathbb{N}$ and $n \ge 0$ then

$$E[(M_n)^{\underline{j}}] = p^n(qn^{\underline{j}} + pE[(M_n)^{\underline{j}}]) + q^n(pn^{\underline{j}} + qE[(M_n)^{\underline{j}}]) + \sum_{k=1}^{n-1} \binom{n}{k} p^k q^{n-k} (pE[(M_k)^{\underline{j}}] + qE[(M_{n-k})^{\underline{j}}]).$$

We derive an asymptotic solution for these recurrence relations using poissonization, the Mellin transform, and depoinssonization (cf. [9], [10]). These methods yield the following two theorems.

Theorem 2 *Let $z_k = \frac{2kr\pi i}{\ln p}$ for all $k \in \mathbb{Z}$, where $\frac{\ln p}{\ln q} = \frac{r}{s}$ for some relatively prime $r, s \in \mathbb{Z}$ (recall that we are interested in the situation where $\frac{\ln p}{\ln q}$ is rational). Then*

$$E[(M_n)^{\underline{j}}] = \Gamma(j)\frac{q(p/q)^j + p(q/p)^j}{h} + \delta_j(\log_{1/p} n) - \frac{1}{2}n\left(\frac{d^2}{dz^2}\delta_j(\log_{1/p} z)\right)\bigg|_{z=n} + O(n^{-2})$$

where

$$\delta_j(t) = \sum_{k \ne 0} -\frac{e^{2kr\pi i t}\Gamma(z_k + j)\left(p^j q^{-z_k - j+1} + q^j p^{-z_k - j + 1}\right)}{p^{-z_k+1}\ln p + q^{-z_k+1}\ln q}$$

and Γ is the Euler gamma function.

We emphasize that the term $-\frac{1}{2}n\left(\frac{d^2}{dz^2}\delta_j(\log_{1/p} z)\right)\Big|_{z=n}$ is $O(n^{-1})$. Note that δ_j is a periodic function that has small magnitude and exhibits fluctuation as shown below for $p = 1/2$.

j	$\frac{1}{\ln 2}\sum_{k\ne 0}\left\lvert\Gamma\left(j - \frac{2ki\pi}{\ln 2}\right)\right\rvert$
1	1.4260×10^{-5}
2	1.3005×10^{-4}
3	1.2072×10^{-3}
4	1.1527×10^{-2}
5	1.1421×10^{-1}
6	1.1823×10^{0}
7	1.2853×10^{1}
8	1.4721×10^{2}

We note that, if $\ln p/\ln q$ is irrational, then $\delta_j(x) \to 0$ as $x \to \infty$. So δ_j does not exhibit fluctuation when $\ln p/\ln q$ is irrational.

The next result describes the asymptotic distribution of M_n.

Theorem 3 *Let $z_k = \frac{2kr\pi i}{\ln p}$ for all $k \in \mathbb{Z}$, where $\frac{\ln p}{\ln q} = \frac{r}{s}$ for some relatively prime $r, s \in \mathbb{Z}$. Then*

$$E[u^{M_n}] = -\frac{q\ln(1-pu) + p\ln(1-qu)}{h} + \delta(\log_{1/p} n, u) - \frac{1}{2}n\left(\frac{\partial^2}{\partial z^2}\delta(\log_{1/p} z, u)\right)\bigg|_{z=n} + O(n^{-2}) \quad (2)$$

where
$$\delta(t,u) = \sum_{k\neq 0} - \frac{e^{2kr\pi it}\Gamma(z_k)\left(q(1-pu)^{-z_k} + p(1-qu)^{-z_k} - p^{-z_k+1} - q^{-z_k+1}\right)}{p^{-z_k+1}\ln p + q^{-z_k+1}\ln q}.$$

and Γ *is the Euler gamma function. It follows immediately that*

$$E[u^{M_n}] = \sum_{j=1}^{\infty}\left[\frac{p^jq+q^jp}{jh} + \sum_{k\neq 0} - \frac{e^{2kr\pi i\log_{1/p}n}\Gamma(z_k)(p^jq+q^jp)(z_k)^{\overline{j}}}{j!(p^{-z_k+1}\ln p + q^{-z_k+1}\ln q)}\right]u^j + O(n^{-1}) \tag{3}$$

and
$$P(M_n = j) = \frac{p^jq+q^jp}{jh} + \sum_{k\neq 0} - \frac{e^{2kr\pi i\log_{1/p}n}\Gamma(z_k)(p^jq+q^jp)(z_k)^{\overline{j}}}{j!(p^{-z_k+1}\ln p + q^{-z_k+1}\ln q)} + O(n^{-1}). \tag{4}$$

If $\ln p/\ln q$ is irrational and u is fixed, then $\delta(x,u) \to 0$ as $x \to \infty$. Thus δ does not exhibit fluctuation when $\ln p/\ln q$ is irrational.

References

[1] Atallah, M. J., and Lonardi, S., Authentication of LZ-77 compressed data. In *Proceedings of the ACM Symposium on Applied Computing* (Melbourne, FL, March 2003), to appear.

[2] Bell, T. C., Cleary, J. G., and Witten, I. H., *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[3] Jacquet, P. and Szpankowski, W., Autocorrelation on words and its applications: Analysis of suffix trees by string-ruler approach, *J. Combinatorial Theory*, A65, 237–269, 1994.

[4] Lonardi, S., Szpankowski, W., Joint source-channel LZ'77 coding, *Data Compression Conference*, 273–282, Snowbird, 2003.

[5] Reed, I. S., and Solomon, G., Polynomial codes over certain finite fields, *J. SIAM*, 8, 300–304, 1960.

[6] Reznik, Y., and Szpankowski, W., On average redundancy rate of the Lempel-Ziv codes with k-error protocol, *Information Sciences*, 135, 57–70, 2001.

[7] Sayood, K., Otu, H., and Demir, N., Joint source/channel coding for variable length codes, *IEEE Trans. Commun.*, 48, 787–794, 2000.

[8] Storer, J., and Reif, J., Error resilient optimal data compression, *SIAM J. Computing*, 26, 934–949, 1997.

[9] Szpankowski, W., *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York, NY, 2001.

[10] Ward, M. D., Analysis of a randomized selection algorithm, M. S. thesis, University of Wisconsin–Madison, 2003.

[11] Ward, M. D., Szpankowski, W., Analysis of a randomized selection algorithm motivated by the LZ'77 scheme. To appear in *1st Workshop on Analytic Algorithms and Combinatorics (ANALCO)*, 2004.

[12] Wyner, A. J., The redundancy and distribution of the phrase lengths of the fixed-database Lempel-Ziv algorithm, *IEEE Trans. Information Theory*, 43, 1439–1465, 1997.

[13] Ziv, J., and Lempel, A., A universal algorithm for sequential data compression, *IEEE Trans. Information Theory*, 23, 337–343, 1977.