LEMPEL-ZIV SLIDING WINDOW UNIVERSAL COMPRESSION

6.441 Supplementary Notes 2b, Revised 3/14/94

INTRODUCTION: In the 70's, there were a number of attempts to develop data compression theories and techniques that could deal with sources with unknown statistics. The most successful of these (both from the standpoint of practice and theory) were two classes of algorithms developed by Jacob Ziv and Abraham Lempel [ZL77, ZL78]. The first scheme, often called LZ77, was a string matching sliding window scheme; the second, called LZ78, was an adaptive dictionary scheme.

The second scheme was implemented many years ago in the UNIX compress algorithm and many other places. Implementations of the first scheme are more recent (Stacker and Microsoft MS-DOS-6); they are viewed as compressing more efficiently, but are more computationally intensive. A new paper, [WZ93], about LZ77 not only analyzes the string matching algorithm but also provides great insight into why universal algorithms work. This note follows [WZ93] closely to explain these issues.

A universal data compression algorithm is an algorithm that runs without any knowledge of the source other than the alphabet. Thus, for a source with known statistics, a universal algorithm cannot compress data any better than an algorithm optimized to those source statistics. We already know that, for any ergodic source with entropy H_∞ and any $\varepsilon > 0$, it is possible to construct fixed to variable length codes (using complete knowledge of the source statistics) that use at most $H_\infty + \varepsilon$ binary digits per source letter; we also know that no codes of any type exist that use fewer than H_∞ binary digits per source letter.

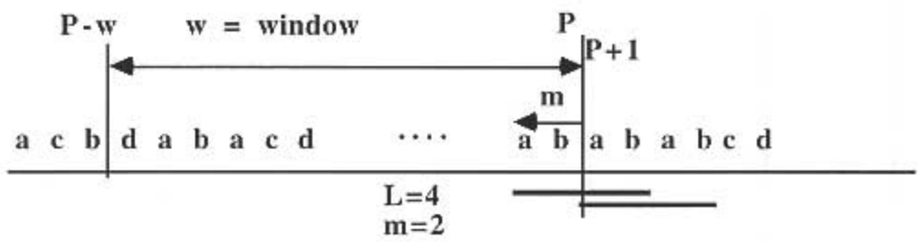
What will be shown here is that LZ77, when applied to any ergodic source, uses at most $H_\infty + \varepsilon$ binary digits per source letter if the window w is sufficiently large. That is, these universal codes can reach the same limit of compressibility as codes using knowledge of the statistics. This is not too surprising, since the statistics can be measured and then used. What is somewhat surprising is that such a simple code is capable of achieving these asymptotic results. Unfortunately, the required window size depends on the statistics of the source, so that in practice, one chooses a fixed window size and hopes for the best.

THE LZ77 ALGORITHM: Given a sequence of letters from a source of alphabet size K, and given a "window size" w which is a power of two:

1) Encode the first w letters without compression, using $\lceil \log K \rceil$ binary digits per letter. (this gets amortized over a very long string; all logs here are base 2)

2) Set pointer P = w (as the algorithm runs, P increases, always dividing the source sequence into its already encoded prefix $x_1^P = x_1, x_2, \ldots, x_P$ and the yet to be encoded sequence $x_{P+1}^{\infty} = x_{P+1}, x_{P+2}, \ldots.$)

3) Find the largest L such that $x_{P+1}^{P+L} = x_{P-m+1}^{P-m+L}$ for some m, $1 \le m \le w$; set L=1 if no such match exists.



Note that the largest match starts two letters inside the window (at m=2) and extends for two letters beyond the window (yielding a match of L=4). Having matches extend beyond the pointer sometimes allows longer matches (as in the present case), and also, as we see later, simplifies the analysis. Note also that there is another match, of length 3, starting one letter from the left end of the window at m=w-1. This is not used, since the longest match is selected. If there are two longest matches, either can be selected.

Figure 1

4) Encode L into a unary-binary code, i.e., $\lfloor \log L \rfloor$ zeros followed by the binary expansion of L:

1 →	1	6 →	00110
2 →	010	7 →	00111
3 →	011	8 →	0001000
4 →	00100	9 →	0001001
5 →	00101	10 →	0001010

(The length of the unary-binary code, for a given L is $2\lfloor \log L \rfloor + 1$; note that the unary-binary code is a fixed to variable length prefix condition coding of the positive integers; see [El75] for a treatment of such integer encodings).

5) If $\lceil \log K \rceil L > \log w$, encode m into log w binary digits; otherwise encode x_{P+1}^{P+L} without compression using $\lceil \log K \rceil$ bits per letter (i.e., $\lceil \log K \rceil L$ bits overall). (One can use the ordinary binary encoding, 1→00....001, 2→00...010, ..., m-1→11...111, and then map m into 00...000).

6) P := P+L; goto step 3 (go on to encode the next match).

COMMENTS ON ALGORITHM: The algorithm is a variable length to variable length encoding. One can regard the dictionary (at any given time) as the set of substrings of the window, plus the single letters of the alphabet. Because of the single letters, the dictionary is valid. On the other hand, the prefix condition is not typically satisfied. The code words are of variable length both because of the unary-binary encoding of L and also because short matches (those with $\lceil \log K \rceil L \le \log w$) are encoded without compression. In particular, the length n of a code word is a function of the length L of the match, and is given by

$$n(L) = 2\lfloor \log L \rfloor + 1 + \min[\log w, \lceil \log K \rceil L] \tag{1}$$

We shall frequently want simple upper bounds to n(L). The first comes from upper bounding $2\lfloor \log L \rfloor + 1$ by $3L/2$ and upper bounding the min term by $\lceil \log K \rceil L$. This yields

$$n(L) \le \gamma L \quad \text{where } \gamma = 3/2 + \lceil \log K \rceil \tag{2}$$

The second bound comes from upper bounding the min term by log w, yielding

$$n(L) \le 2 \log L + 1 + \log w \tag{3}$$

Both bounds are valid for all L, but the first is useful for L small, and the second for L large.

We next explain how a receiver of the encoded sequence can retrieve the source sequence. First the initial window is retrieved from the initial $\lceil \log K \rceil w$ encoded binary digits. The next string of binary digits is the unary-binary encoding of the first match length. This is decoded, using the prefix condition of the encoding, and the receiver then determines if $\lceil \log K \rceil L > \log w$. If so, the position m of the match is decoded, and $\mathbf{x}_{w+1}^{w+L}$ is retrieved by extracting $\mathbf{x}_{w-m+1}^{w-m+L}$ from the stored window; if not $\mathbf{x}_{w+1}^{w+L}$ is decoded directly. This then allows the window to be updated at the receiver, and further decoding proceeds in the same way.

In the case of encoding a finite length string, say $\mathbf{x}_1^N$, step 3 of the algorithm must be slightly modified to stop after the entire string is encoded. That is, step 3 must be modified

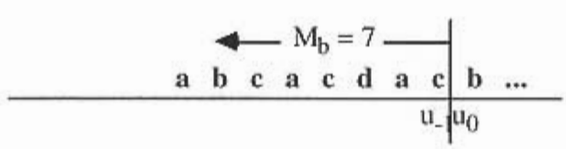
to find the largest $L \le N-P$ such that $\mathbf{x}_{P+1}^{P+L} = \mathbf{x}_{P-m+1}^{P-m+L}$ for some m, $1 \le m \le w$. Also, step 6 must be modified to stop when P is incremented to N.

This code is clearly universal since the algorithm uses no knowledge of source statistics. What we now show is that the code can approach the entropy bound for any ergodic source as w becomes large. Given any window size w, the lengths of successive matches are random variables. We start by looking at the first match length L; intuitively we expect all match lengths to be statistically similar, but unfortunately, the fact that a parse occurs at a particular point effects the distribution of the next match length.

It turns out that when the window w is very large, L is close to $(\log w)/H_\infty$ with high probability. For such values of L, (3) is not only a bound to n(L) but also a good approximation, so $n(L)/L \approx [2 \log L + 1 + \log w]/L$. As $w \to \infty$ (with $L \approx (\log w)/H_\infty$), $n(L)/L$ approaches $(\log w)/L \approx H_\infty$. In the next section, we show that the first match length L is close to $(\log w)/H_\infty$ with high probability, and then in the following section, we show that the effect of the parsing process is negligible.

ANALYSIS OF THE FIRST MATCH LENGTH: Instead of looking directly at the length L of the longest match, we look at a fixed length string $\mathbf{x}_{P+1}^{P+L}$ starting at the pointer and find the expected distance back into the past until that string occurs again. We start with an even simpler problem: let $\{U_{-\infty}^{\infty}\}$ be an ergodic source, let b be a letter in that source, and let M_b be a positive integer valued random variable defined as the negative of the latest time before time 0 at which the letter b occurs. Thus M_b takes on the value $m \ge 1$ if, first,
$U_{-m} = b$ and, second, $U_{-j} \ne b$ for $1 \le j < m$. Thus

$$\Pr(M_b = m \mid U_0 = b) = \Pr\{U_{-m} = b,\ U_j \ne b, -m < j < 0 \mid U_0 = b\} \quad ; m \ge 1 \tag{4}$$



Theorem 1: If $\Pr(U_0=b)>0$, then $E[M_b \mid U_0=b] = 1/\Pr(U_0=b)$

Proof: Define the event $A_{j,k} = \{U_{-j}=b,\ U_i\neq b \text{ for } -j<i<k,\ U_k=b\}$ for each integer $j\geq 1$, $k\geq 0$. $A_{j,k}$ is the event that the most recent occurrence of letter b before time 0 is at -j, and that the first occurrence of letter b at time 0 or later is at k. Thus the events $\{A_{j,k}\}$ are disjoint. Also, since $\Pr(U_0=b)>0$ and the source is ergodic, the letter b must appear somewhere both in the past and future with probability 1. Thus the probabilities of the events $\{A_{j,k}\}$ sum to 1, yielding

$$1 = \sum_{j=1}^{\infty} \sum_{k=0}^{\infty} \Pr(A_{j,k}) = \sum_{j=1}^{\infty} \sum_{k=0}^{\infty} \Pr(A_{j+k,0}) \tag{5}$$

where we have used stationarity to see that $\Pr(A_{j,k}) = \Pr(A_{j+k,0})$. For each $i>0$ the above sum has i distinct but equal terms with $j+k=i$, and thus (5) can be rewritten as

$$1 = \sum_{i=1}^{\infty} i \Pr(A_{i,0}) = \sum_{i=1}^{\infty} i \Pr(M_b=i,\ U_0=b)$$

$$= \Pr(U_0=b) \sum_{i=1}^{\infty} i \Pr(M_b=i \mid U_0=b) = \Pr(U_0=b) E[M_b \mid U_0=b]$$

which is the statement of the theorem.

From the Markov inequality, the probability that b does not occur in the window from -w to -1, conditional on $U_0=b$, satisfies the inequality

$$\Pr(M_b>w \mid U_0=b) \leq \frac{E[M_b \mid U_0=b]}{w} = \frac{1}{w \Pr(U_0=b)} \tag{6}$$

Now for the ergodic source $\{\mathbf{X}_{-\infty}^{\infty}\}$ of interest here, let Λ be a fixed positive integer, and let $U_i = \mathbf{X}_i^{i+\Lambda-1}$. Thus U_i is a Λ-tuple of letters from the X alphabet, and successive letters of $\{\mathbf{U}_{-\infty}^{\infty}\}$ are overlapping Λ-tuples from X. Now let $\mathbf{z}$ be some given Λ-tuple from the X alphabet satisfying $\Pr(\mathbf{X}_1^{\Lambda} = \mathbf{z}) > 0$, and let $M_{\mathbf{z}}$ be a positive integer valued random variable defined as the negative of the latest starting time before 0 at which the Λ-tuple $\mathbf{z}$ occurs. Thus, as in (4),

$$\Pr(M_{\mathbf{z}}=m \mid \mathbf{X}_1^{\Lambda} = \mathbf{z}) = \Pr\{\mathbf{X}_{1-m}^{\Lambda-m}=\mathbf{z},\ \mathbf{X}_{1-j}^{\Lambda-j}\neq\mathbf{z},\ -m<j<0 \mid \mathbf{X}_1^{\Lambda}=\mathbf{z}\} \quad ;\ m\geq 1 \tag{7}$$

Since $\{\mathbf{X}_{-\infty}^{\infty}\}$ is ergodic, $\{\mathbf{U}_{-\infty}^{\infty}\}$ is ergodic also, and thus it follows from theorem 1 that

$$E[M_x \mid \mathbf{X}_1^{\Lambda}=\mathbf{z}] = \frac{1}{\Pr(\mathbf{X}_1^{\Lambda}=\mathbf{z})} \tag{8}$$

Applying the Markov inequality as before,

$$\Pr[M_x>w \mid \mathbf{X}_1^{\Lambda}=\mathbf{z}] \le \frac{1}{w\Pr(\mathbf{X}_1^{\Lambda}=\mathbf{z})} \tag{9}$$

Note that, given $\mathbf{X}_1^{\Lambda} = \mathbf{z}$, the event $M_z>w$ is the event that no match to $\mathbf{X}_1^{\Lambda}$ exists within the window of size w. Thus, letting L_w be the length of the longest match to $\mathbf{X}_1^{\infty}$ in the window from -w+1 to 0,

$$\Pr(L_w<\Lambda \mid \mathbf{X}_1^{\Lambda}=\mathbf{z}) = \Pr[M_x>w \mid \mathbf{X}_1^{\Lambda}=\mathbf{z}] \le \frac{1}{w\Pr(\mathbf{X}_1^{\Lambda}=\mathbf{z})} \tag{10}$$

Averaging over $\mathbf{X}_1^{\Lambda}$ then yields

$$\Pr[L_w<\Lambda] = \textstyle\sum_{\mathbf{z}} \Pr(M_z>w \mid \mathbf{X}_1^{\Lambda} = \mathbf{z}) \Pr(\mathbf{X}_1^{\Lambda} = \mathbf{z}). \tag{11}$$

ASYMPTOTIC BEHAVIOR: Our next objective is to find an upper bound to $\Pr[L_w<\Lambda]$ that approaches 0 for large Λ and w. Specifically, we set $\Lambda = \lfloor(\log w)/[H_\infty+\varepsilon]\rfloor$ for some fixed $\varepsilon>0$ and go the limit $w\to\infty$. We first review the asymptotic equipartition property of information theory (see text, section 3.5).

For some given $\delta>0$, define the typical set of Λ-tuples $\mathbf{z}$ as

$$T_\Lambda(\delta) = \left\{\mathbf{z} : \left| -\frac{1}{\Lambda}\log[\Pr(\mathbf{X}_1^{\Lambda}=\mathbf{z}] - H_\infty \right| < \delta\right\} \tag{12}$$

Then the AEP asserts that

$$\lim_{\Lambda\to\infty}\Pr(\mathbf{X}_1^{\Lambda} \in T_\Lambda(\delta)) = 1 \tag{13}$$

That is, for any $\delta>0$, the aggregate probability of atypical Λ-tuples is negligible for sufficiently large Λ. From (12), we also see that typical sequences must have typical individual probabilities,

$$2^{-\Lambda(H_\infty+\delta)} < \Pr(\mathbf{X}_1^{\Lambda}=\mathbf{z}) \le 2^{-\Lambda(H_\infty-\delta)} \;;\; \text{for all } \mathbf{z}\in T_\delta(\Lambda) \tag{14}$$

and thus the size of the typical set satisfies

$$|T_\Lambda(\delta)| \le 2^{\Lambda[H_\infty+\delta]} \tag{15}$$

Theorem 2: Let $\Lambda_w = \lfloor(\log w)/[H_\infty+\varepsilon]\rfloor$. Then

$$\lim_{w\to\infty} \Pr(L_w<\Lambda_w) = 0$$

Proof: Splitting the sum in (11) into two terms, one for $\mathbf{z}\in T_{\Lambda_w}(\delta)$ and the other for $\mathbf{z}\notin T_{\Lambda_w}(\delta)$,

$$\Pr[L_w<\Lambda_w] = \textstyle\sum_{\mathbf{z}\in T_{\Lambda_w}(\delta)} \Pr(\mathbf{X}_1^{\Lambda_w} = \mathbf{z})\Pr(L_w<\Lambda_w|\mathbf{X}_1^{\Lambda_w} = \mathbf{z})$$

$$+ \textstyle\sum_{\mathbf{z}\notin T_{\Lambda_w}(\delta)} \Pr(\mathbf{X}_1^{\Lambda_w} = \mathbf{z})\Pr(L_w<\Lambda_w|\mathbf{X}_1^{\Lambda_w} = \mathbf{z})$$

$$\le \sum_{\mathbf{z}\in T_{\Lambda_w}(\delta)} \Pr(\mathbf{X}_1^{\Lambda_w}=\mathbf{z}) \frac{1}{w\Pr(\mathbf{X}_1^{\Lambda_w}=\mathbf{z})} + \sum_{\mathbf{z}\notin T_{\Lambda_w}(\delta)} \Pr(\mathbf{X}_1^{\Lambda_w}=\mathbf{z})$$

where we have bounded $\Pr(L_w<\Lambda_w|\mathbf{X}_1^{\Lambda_w} = \mathbf{z})$ by (10) in the first sum and by 1 in the second. Since each term in the first sum is now simply $1/w$, the first sum is $|T_{\Lambda_w}(\delta)|/w$, so

$$\Pr(L_w<\Lambda_w) \le \frac{|T_{\Lambda_w}(\delta)|}{w} + \Pr(\mathbf{X}_1^{\Lambda_w}\notin T_{\Lambda_w}(\delta))$$

$$\le \frac{2^{\Lambda_w[H_\infty+\delta]}}{w} + \Pr(\mathbf{X}_1^{\Lambda_w}\notin T_\Lambda(\delta) \tag{16}$$

where we have used (15). Since $\Lambda_w = \lfloor(\log w)/[H_\infty+\varepsilon]\rfloor \le (\log w)/[H_\infty+\varepsilon]$, we have $w \ge 2^{\Lambda_w(H_\infty+\varepsilon)}$. Using this bound in (16) yields

$$\Pr(L_w<\Lambda_w) \le 2^{\Lambda_w[\delta-\varepsilon]} + \Pr(\mathbf{X}_1^{\Lambda_w}\notin T_{\Lambda_w}(\delta)) \tag{17}$$

Choosing $\delta = \varepsilon/2$, and noting that $\Lambda_w \to \infty$ as $w \to \infty$, we have $\lim_{w\to\infty}\Pr(L_w<\Lambda_w) = 0$, completing the proof.

THE EFFECT OF PARSING: We can not use (17) directly in analyzing LZ77, since the fact that a parse occurs at point P in the source sequence effects the size of the next match. Instead, we will complete the analysis by comparing the operation of LZ77 on a given string of source letters with the operation of a fixed to variable length algorithm. For given w and $\Lambda_w = \lfloor(\log w)/[H_\infty+\varepsilon]\rfloor$, let $\mathbf{x}_1^\infty$ be a given sequence of source letters and consider parsing $\mathbf{x}_1^\infty$ into an initial segment of length w followed by subsequent fixed length segments, $\mathbf{z}(1)$, $\mathbf{z}(2)$,... of length Λ_w (see Figure 2).

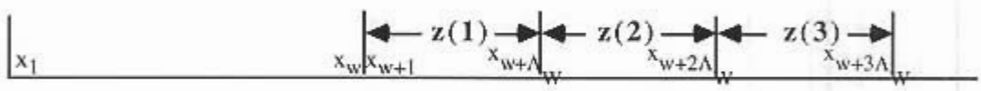


Figure 2

Initially restrict attention to a finite string, say $\mathbf{x}_1^{w+N\Lambda_w}$ of source letters. After the initial window, this string is partitioned into the fixed length segments $\mathbf{z}(1)$, $\mathbf{z}(2)$,...,$\mathbf{z}(N)$. This same string (after the initial window) is also partitioned into the segments $\mathbf{y}(1)$, $\mathbf{y}(2)$,...,$\mathbf{y}(M)$ for some M, where $\mathbf{y}(k)$ is the kth segment of $\mathbf{x}_1^{w+N\Lambda_w}$ found by LZ77, modified as in the comments on the algorithm to stop after encoding the entire string $\mathbf{x}_1^{w+N\Lambda_w}$. In what follows, we refer to $\mathbf{z}(1)$, ..., $\mathbf{z}(N)$ as fixed length (FL) segments, and refer to $\mathbf{y}(1)$, ..., $\mathbf{y}(M)$ as LZ segments (see Figure 3).

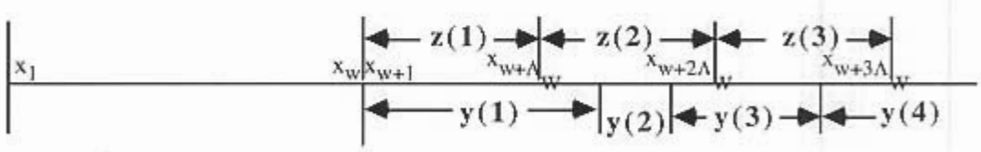


Figure 3

For any segment $\mathbf{x}_i^j$ of $\mathbf{x}_1^{w+N\Lambda_w}$, $w<i<j$, we say that $\mathbf{x}_i^j$ has a match if $\mathbf{x}_i^j = \mathbf{x}_{i-m}^{j-m}$ for some m, $1\le m\le w$. From theorem 2, if $\mathbf{x}_1^{w+N\Lambda_w}$ is a sample sequence from the ergodic source under consideration, and w is large, then there is a high probability that $\mathbf{z}(n)$ has a match

for any given n, $1 \le n \le N$. The following simple lemma lets us relate $\{z(n);\ 1 \le n \le N\}$ to $\{y(k);\ 1 \le k \le M\}$.

Lemma: If x_i^j has a match, then each sub-segment of x_i^j has a match.

Proof: If x_i^j has a match, then $x_i^j = x_{i-m}^{j-m}$ for some m, $1 \le m \le w$, which means that $x_k = x_{k-m}$ for $i \le k \le j$. Thus, for any sub-segment $x_{i'}^{j'}$, $i' \ge i$, $j' \le j$, $x_k = x_{k-m}$ for $i' \le k \le j'$ and $x_{i'}^{j'}$ has a match.

To use this lemma, suppose that the FL segment $z(n) = x_i^j$ has a match and that the LZ segment $y(k) = x_{i'}^{j'}$ starts within the segment z(n) (i.e., $i \le i' \le j$). Then we claim that y(k) extends at least to the end of z(n) (i.e., that $j' \ge j$). To see this, note that $x_{i'}^{j}$ (i.e., the segment starting at the beginning of y(k) and ending at the end of z(n)) must have a match since it is a sub-segment of z(n). Since y(k) is the longest match starting at i', y(k) extends at least to the end of z(n).

The above argument shows that each FL segment z(n) with a match has at most one LZ segment y(k), $1 \le k \le M$, starting within it. Figure 4 illustrates this.

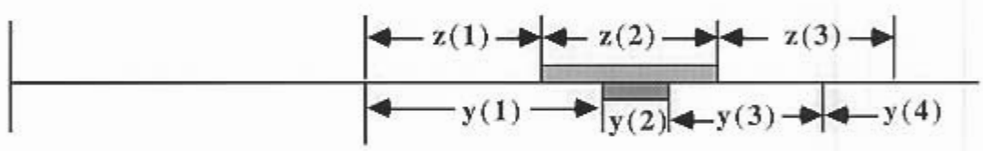


Figure 4
Each segment z(n) with a match (z(1) and z(3) in the example) has at most one segment y(k) starting within it (y(1) and y(4) respectively in the example); each segment without a match (z(2) in the example, indicated by crosshatching) might have several segments (y(2) and y(3) above) starting within it.

Now define a LZ segment y(k) to be "bad" if it is contained within some FL segment z(n), $1 \le n \le N$. We saw above that segments z(n) with matches cannot contain bad LZ segments y(k). The segment y(2) in figure 4 is bad, and is contained in z(2), which does not have a match. Define B as the set of segment numbers of bad LZ segments y(k).

$$B = \{k : y(k) \text{ is contained within } z(n) \text{ for some } n,\ 1 \le n \le N\} \tag{18}$$

Let $L_1, L_2, \ldots, L_M$ be the lengths of $y(1), \ldots, y(M)$ and let ϕ be the fraction of the FL segments $z(1), \ldots, z(N)$ that do not have matches. Note that $\sum_{k \in B} L_k$ is the total number of

positions in $\mathbf{x}_1^{w+N\Lambda_w}$ that are contained in bad LZ segments, and thus all of these positions are also contained in FL segments $\mathbf{z}(n)$ that do not have matches. Since there are ϕN FL segments $\mathbf{z}(n)$ without matches, we have

$$\sum_{k\in B} L_k \le N\phi\Lambda_w \tag{19}$$

From (2), we can upper bound the number of code digits for these segments by

$$\sum_{k\in B} n(L_k) \le \gamma N\phi\Lambda_w \quad \text{where } \gamma = 3/2 + \lceil \log K \rceil \tag{20}$$

Also, for each FL segment $\mathbf{z}(n)$ with a match, at most one LZ segment $\mathbf{y}(k)$ can start in $\mathbf{z}(n)$. Similarly, for $\mathbf{z}(n)$ without a match, there can be at most one LZ segment $\mathbf{y}(k)$ that starts in $\mathbf{z}(n)$ and is not contained in $\mathbf{z}(n)$ (i.e., $\mathbf{y}(3)$ in figure 4). Thus the number of LZ segments $\mathbf{y}(k)$ excluding bad segments is at most N, so $|B^c| \le N$. We can now find an upper bound on the sum of the code word lengths for the LZ segments $\mathbf{y}(k)$ that are not bad. For $k \in B^c$, we use (3) to upper bound $n(L_k)$ by $1+\log w + 2\log(L_k)$. Thus

$$\begin{aligned}
\sum_{k\in B^c} n(L_k) &\le (1+\log w)N + \sum_{k\in B^c} 2\log(L_k) \\
&= (1+\log w)N + |B^c| \sum_{k\in B^c} \frac{2\log(L_k)}{|B^c|} \\
&\le (1+\log w)N + 2|B^c| \left[\log \sum_{k\in B^c} \frac{(L_k)}{|B^c|} \right]
\end{aligned}$$

where we have used Jensen's inequality. Since this quantity is increasing in $|B^c|$, we can further bound by replacing $|B^c|$ by N. We can also further bound by summing L_k over all k. Thus

$$\sum_{k\in B^c} n(L_k) \le (1+\log w)N + 2N\log(\Lambda_w) \tag{21}$$

Combining (20) and (21), and dividing by $N\Lambda_w$ to obtain the number of binary digits per source letter,

$$\sum_{k=1}^{M} \frac{n(L_k)}{N\Lambda_w} \le \frac{1+\log w}{\Lambda_w} + \frac{2\log(\Lambda_w)}{\Lambda_w} + \gamma\phi \tag{22}$$

Since $\Lambda_w = \lfloor (\log w)/[H_\infty+\varepsilon] \rfloor \ge (\log w)/[H_\infty+\varepsilon] - 1$, we can simplify this to

$$\sum_{k=1}^{M} \frac{n(L_k)}{N\Lambda_w} \le \frac{1+\log w}{(\log w)/(H_\infty+\varepsilon)-1} + \frac{2\log(\Lambda_w)}{\Lambda_w} + \gamma\phi$$

$$= \frac{(H_\infty+\varepsilon)(1+1/\log w)}{1-(H_\infty+\varepsilon)/\log w} + \frac{2\log(\Lambda_w)}{\Lambda_w} + \gamma\phi$$

We now take the expected value of this and then pass to the limit $w\to\infty$. The first term goes to the limit $H_\infty+\varepsilon$. The second term goes to 0 in the limit, since $\Lambda_w \to\infty$. In the third term, $E[\phi] = \Pr[L_w<\Lambda_w]$, and this goes to 0 as $w\to\infty$ by theorem 2. Thus,

$$\lim_{w\to\infty} E\left[\sum_{k=1}^{M} \frac{n(L_k)}{N\Lambda_w}\right] \le H_\infty+\varepsilon \tag{23}$$

This yields the desired bound on the expected number of encoded binary digits per source letter. We must also go the limit $N\to\infty$ in order to amortize the effect of the initial window.

The analysis we have gone through provides us with some additional insights. First, if we know H_∞ and know how large Λ must be for $T_\Lambda(\delta)$ to act like a typical set, then we could use the segments $\mathbf{z}(1)$, $\mathbf{z}(2)$,... of the last section as a fixed length to variable length code. We would simply choose Λ_w to be sufficiently large, and then choose $w = 2^{\Lambda_w(H_\infty+\varepsilon)}$. According to our analysis, the window would then contain the typical strings of length Λ_w with high probability, and the scheme would compress the source to close to H_∞ binary digits per source digit.

We could visualize this fixed to variable length scheme as having a dictionary consisting of the typical sequences in the window. Note that for a fixed to variable length code, we usually regard the dictionary size as K^Λ where Λ is the block length of source. Here we regard the dictionary size as $2^{\Lambda H_\infty}$ and view the nontypical strings as strings that are encoded letter by letter. Thus we see that the very large dictionaries required by fixed to variable length codes are simply required by the huge number of atypical, improbable strings; handling these improbable strings in the simpler way here is clearly preferable.

REFERENCES:

[El75] Elias, P., "Universal Code Word Sets and Representations of the Integers," IEEE Trans. IT, March 1975, pp.194-203.

[LZ76] Lempel, Abraham & Jacob Ziv, "On the Complexity of Finite Sequences," IEEE Trans. IT, Jan. 1976, pp. 75-81.

[St88] Storer, James A., *Data Compression, methods and theory,* Computer Science Press 1988.

[WZ93] Wyner, Aaron & Jacob Ziv, "The Sliding Window Lempel-Ziv Algorithm is Asymptotically Optimal," Preprint

[ZL77] Ziv, Jacob & Abraham Lempel, "A Universal Algorithm for sequential data compression," IEEE Trans. IT, May 1977, pp. 337-343.

[ZL78] Ziv, Jacob & Abraham Lempel, "Compression of Individual Sequences via Variable-Rate Coding," IEEE Trans. IT, Sept. 1978, PP. 530-536.