Title
Error resilient LZ'77 data compression: Algorithms, analysis, and experiments

Journal
IEEE Transactions on Information Theory, 53(5)

ISSN
0018-9448

Authors
Szpankowski, Wojciech
Lonardi, Stefano
Ward, Mark Daniel

Publication Date
2007-05-01

Peer reviewed

Error Resilient LZ'77 Data Compression: Algorithms, Analysis, and Experiments

Stefano Lonardi, *Member, IEEE*, Wojciech Szpankowski, *Fellow, IEEE*, and Mark Daniel Ward, *Member, IEEE*

***Abstract*—We propose a joint source–channel coding algorithm capable of correcting some errors in the popular Lempel–Ziv'77 (LZ'77) scheme without introducing any measurable degradation in the compression performance. This can be achieved because the LZ'77 encoder does not completely eliminate the redundancy present in the input sequence. One source of redundancy can be observed when an LZ'77 phrase has multiple matches. In this case, LZ'77 can issue a pointer to any of those matches, and a particular choice carries some additional bits of information. We call a scheme with embedded redundant information the LZS'77 algorithm. We analyze the number of longest matches in such a scheme and prove that it follows the *logarithmic series* distribution with mean $1/h$ (plus some fluctuations), where h is the source entropy. Thus, the distribution associated with the number of redundant bits is well concentrated around its mean, a highly desirable property for error correction. These analytic results are proved by a combination of combinatorial, probabilistic, and analytic methods (e.g., Mellin transform, depoissonization, combinatorics on words). In fact, we analyze LZS'77 by studying the *multiplicity matching parameter* in a suffix tree, which in turn is analyzed via comparison to its independent version, called *trie*. Finally, we present an algorithm in which a channel coder (e.g., Reed–Solomon (RS) coder) succinctly uses the inherent additional redundancy left by the LZS'77 encoder to detect and correct a limited number of errors. We call such a scheme the LZRS'77 algorithm. LZRS'77 is perfectly backward-compatible with LZ'77, that is, a file compressed with our error-resistant LZRS'77 can still be decompressed by a generic LZ'77 decoder.**

***Index Terms*—Autocorrelation polynomial, combinatorics on words, depoissonization, joint source–channel coding, Lempel–Ziv'77 (LZ'77) scheme, Mellin transform, multiple matches, pattern matching, Reed–Solomon (RS) code, suffix trees, tries.**

Manuscript received May 26, 2005; revised January 15, 2007. The work of S. Lonardi was supported in part by the National Science Foundation under Grant DBI-0321756 a nd under NSF CAREER IIS-0447773. The work of W. Szpankowski was supported in part by the National Science Foundation under Grant CCR-0208709, the National Institutes of Health under Grant R01 GM068959-01, and by AFOSR under Grant FA8655-04-1-3074. The work of M. D. Ward was supported by the National Science Foundation under Grant 0603821. The material in this paper were presented in part at the Data Compression Conference, Snowbird, UT, March 2003 and the IEEE International Symposium on Information Theory, Chicago, IL, June/July 2004.

S. Lonardi is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92521 US (e-mail: stelo@cs.ucr.edu).

W. Szpankowski is with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907 USA (e-mail: spa@cs.purdue.edu).

M. D. Ward is with the Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: ward2@math.upenn.edu).

Communicated by M. Effros, Associate Editor for Source Coding.

Color versions of Figures 1, 3, and 7 in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2007.894689

I. 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 [25]. In fact, in many applications, a practical drawback of adaptive data compression algorithms is their lack of resistance to errors. Joint source–channel coding has emerged as a viable solution to this problem.

The *separation principle* formulated by Shannon divides a communication system into separate source coding and channel coding subsystems that run independently; however, in today's communication technology this rigid separation is very limiting. In particular, this principle ignores many imperfections of real communication systems, such as the fact that channel coding is incapable of correcting all errors. Uncorrectable errors are inevitable; designing encoders while ignoring this fact simply leads to extremely fragile source codes, in which one single error can potentially yield catastrophic failures. Joint source–channel coding strikes a balance between source bits versus channel bits, which in turn requires some adjustments in both the source coding and channel coding strategies. Our approach is somewhat orthogonal to most works in this area. We use redundancy bits left by the source coder to protect against errors *without degrading the compression rate*. The price we pay is that we only correct a few errors, and we do not achieve a positive error bit rate (i.e., we are unable to correct a number of errors proportional to the size of a block). We do not address here error propagation (cf. [25]); however, by eliminating errors, our algorithm implicitly protects against limited error propagation.

In this paper, we deal with one of the best known adaptive data compression schemes, namely that of Ziv and Lempel published in their 1977 seminal paper [33]. The popular LZ'77 compression scheme works online. It compresses phrases by consecutively replacing the longest prefix of the noncompressed portion of a file with a *pointer* and the *length* of the prefix. The lack of error resistance of LZ'77 is a well-recognized problem. A few years ago we read the following posting on the `comp.compression` newsgroup: "...I'm a casualty of corrupt tar'd[1] gzipped files on Solaris 8. (`gzip 1.3`) ... Is

[1] `tar` is a common archiver under the Unix operating system.

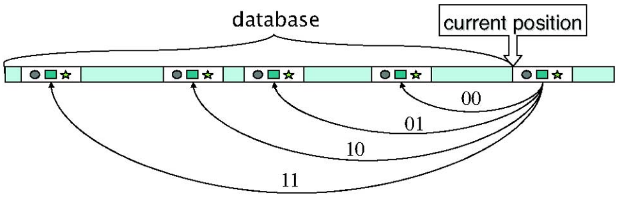

Fig. 1. The multiplicity of the next phrase is four ($M = 4$). Choosing one of the four possible pointers recovers two redundant bits.

there a reason why there are no compression utilities that allow controlled amounts of redundancy for error correction? ... How much overhead would be needed to correct these?"

Indeed, we asked ourselves, how much overhead is needed in LZ'77 to correct errors? The surprising answer is that there is no need for additional overhead in order to correct some errors in LZ'77. This seemingly impossible goal is achieved in practice thanks to the fact that the LZ'77 encoder is unable to completely decorrelate the input sequence. Some implicit redundancy, which we precisely quantify in this paper, 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 $M > 1$ possible pointers. In practice, if there are M copies of the longest prefix, we recover $\lfloor \log_2 M \rfloor$ redundant bits by choosing one of the M pointers (see Fig. 1). We call such a scheme with multiple pointers the LZS'77 algorithm.

In the first part of the paper, we present an algorithm for channel coding that exploits the redundant bits identified by LZS'77. To detect and correct errors, we choose Reed–Solomon (RS) codes computed on blocks of 255 bytes of compressed data. Given the maximum number of errors e that the RS code can correct, the $2e$ parity bits of the RS code will be embedded in the extra redundant bits extracted from the pointer multiplicity. We should point out that if e is large then we may not always have enough redundant bits to embed the parity bits. The algorithm that incorporates the RS channel coding into LZS'77 is referred to throughout as the LZRS'77 scheme.

As mentioned earlier, our basic algorithm allows one to correct only a few errors, thus we set $e = O(1)$, and e is rather small in our implementations. In fact, we prove theoretically that asymptotically the average number of longest phrases is $O(1)$ leading to $e = O(1)$. We should observe, however, that even single errors can have devastating effects. It has been proved recently [4] that a single error in LZ'77 may corrupt up to $O(n^{2/3})$ phrases, thus about $O(n^{2/3} \log n)$ symbols, where n is the size the file to be compressed. Furthermore, a simple modification of our algorithm (e.g., instead of looking for the longest match we just consider a "long enough" match) allows e to change adaptively with the availability of redundancy bits in the stream (i.e., e will slowly grow with n) and still preserve the asymptotic optimality of the compression bit rate (see Remark (i) after Theorem 1).

In the second part of this paper, we theoretically quantify the amount of redundancy left by the LZ'77 encoder for error protection. Thus, we resort to analyzing the number of pointers in the LZS'77 schemes, a problem never addressed before. We let M_n denote the *number of pointers* (longest matches) into the database when n bits have already been compressed. We are primarily interested in precisely determining the asymptotics of the random variable M_n and its concentration around the mean. A thorough analysis of the variable M_n yields a characterization of the degree to which error correction can be performed in the scheme discussed above. We recall that $\lfloor \log_2 M_n \rfloor$ bits are available for detecting and correcting errors.

Suffix trees provide a natural way to study the variable M_n. A suffix tree [27] is a digital search tree (i.e., a *trie* [27]) built from all the suffixes of a single string (the database in our case). In a suffix tree, M_n corresponds to the number of leaves in the subtree rooted at the branching point of the $(n+1)$th insertion. We refer to M_n as the *multiplicity matching parameter*. As it turns out, strings in suffix trees are highly dependent on each other. This dependency complicates the precise analysis of M_n; therefore, we also consider the analogous situation, where a trie is built over independent strings. More specifically, we study the variable M_n^I associated with the number of leaves in the subtree rooted at the branching point of the $(n+1)$th insertion in a trie. After determining the asymptotics of M_n^I, we prove that M_n and M_n^I have asymptotically identical distributions.

The main theoretical result consists of a precise characterization of all the moments of M_n and its limiting distribution. In particular, we show that for memoryless sources,[2] the average number of pointers is $1/h$, where h is the entropy rate. We also show that the limiting distribution of M_n follows the *logarithmic series distribution*, that is

$$\Pr(M_n = k) \approx (p^k(1-p) + (1-p)^k p)/(kh)$$

where p is the probability of generating a "1." Thus, the number of pointers is well concentrated around the mean, which is a highly desirable property for channel coding. Still, it is more likely to have one occurrence of the longest phrase in the database than many, but the probability of seeing two longest phrases is only four times smaller than finding a single longest phrase. In practice, we usually find more than one match, as shown in Section II.

In order to prove our main result we use a battery of analytic tools, including analytical poissonization and depoissonization, the Mellin transform, and complex analysis. To prove that suffix trees and independent tries have similar multiplicity matching parameters, we derive bivariate generating functions for M_n and M_n^I using combinatorics on words, as recently surveyed in [17]. We compare the generating functions for M_n and M_n^I by utilizing complex asymptotics.

To the best of our knowledge, the scheme described here is the first joint source–channel LZ'77 algorithm. In [25], Storer and Reif address the issue of *error propagation* but not error recovery (see [21] 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., [23]). Recently, we have proposed a novel scheme to extract redundant bits from LZ'78/LZW streams [31].

Regarding our theoretical results, the multiplicity matching parameter was never previously studied in tries and suffix trees. However, the methodology used here to study the matching

[2]Our analysis can be extended to Markov sources using the techniques developed in this paper.

parameter *in tries* is well established within the analytic algorithmic community [27]. The analysis of M_n in a suffix tree is new and quite challenging. The basic idea of comparing suffix trees to independent tries was established by Jacquet and Szpankowski [11] and recently simplified by these authors in [17]. Other aspects of suffix trees have been studied in [5], [7], [26].

The paper is organized as follows. In Section II-A, we describe the LZS'77 encoder and present our main theoretical results. In Section II-B, we design the encoder and decoder for the LZRS'77 scheme and in Section II-C discuss the experiment results. The main theoretical result is proved in Sections III–V. In Section III, we provide a streamlined analysis and the roadmap of the proof. Independent tries are discussed in Section IV while suffix trees are analyzed in Section V.

II. Main Results

In this section, we present our main algorithmic, theoretical, and experimental results. We first describe a modified LZ'77 scheme, called LZS'77, in which we recover redundant information by identifying multiple longest matches. In Theorem 1, we quantify the redundant information by analyzing the variable M_n, associated with the number of longest matches when the database sequence is of length n. Finally, the recovered redundant bits are used in a new algorithm called LZRS'77, in which $O(1)$ errors are corrected at each stage of the compression. We end the section by reporting experimental results on LZRS'77.

A. Redundant Information in LZS'77

Let X be a text of length n over a finite alphabet $\mathcal{A}$. We write X_i, $1 \le i \le n$ to indicate the ith symbol in X. We use X_i^j as shorthand for the substring $X_i X_{i+1} \cdots X_j$, where $1 \le i \le j \le n$, with the convention that $X_i^i = X_i$. Substrings of the form X_1^j correspond to prefixes of X, and substrings of the form X_i^n correspond to the suffixes of X.

The LZ'77 algorithm [33] processes the data *online* as it is read, i.e., it parses the file sequentially *left to right* and looks into the sequence of past symbols (called the *database*) to find a match with the longest prefix of the string starting at the current position. The longest prefix is replaced with a *pointer*, which is a triple composed of (*position, length, symbol*). Several variations on LZ'77 have been proposed (see, e.g., [3] and references therein), but the basic principle remains the same.

Let us suppose that the first $i-1$ symbols of the string X have been already parsed into $k-1$ phrases, i.e., $X_1^{i-1} = y_1 y_2 \cdots y_{k-1}$, where each y is a nonempty string over $\mathcal{A}$. In order to identify the kth phrase, LZ'77 looks for the *longest* prefix of X_i^n that matches a substring of X_1^{i-1}. If X_j^{j+l-1} $(j < i)$ is the substring that matches the longest prefix, then the next phrase is $y_k = X_i^{i+l-1}$. The algorithm issues the pointer (j, l, X_{i+l}) and updates the current position i to $i+l+1$. The symbol X_{i+l} is needed to be able to advance when $l = 0$, which is common in the very beginning of the encoding process. The use of a raw symbol within each pointer is wasteful in practice, because it can often be included in the next pointer. Later, we will assume that the LZ'77 compressed stream is just a sequence of (*position, length*) pointers, as it is implemented in `gzip` and other encoders.

```
LZS'77_Encoder (X, K)
let i, r, n, m, P ← 0, 0, |X|, |K|, []
while i < n do
    let X_i^{i+l-1} ← the longest prefix of X_i^t
        that matches a substring in X_1^{i-1}
    let R ← {(p_0, l, X_{i+l}), ..., (p_{M-1}, l, X_{i+l})} be
        the set of feasible pointers for X_i^{i+l-1}
    if M > 1 then
        let d ← ⌊log_2 M⌋
        append (p_{K_r^{r+d}}, l, X_{i+l}) to P
        let r ← r + d
    else
        append (p_{M-1}, l, X_{i+l}) to P
    let i ← i + l + 1
return P

LZS'77_Decoder (P)
let D, K ← empty string, empty string
for each (p, l, c) ∈ P do
    let R ← {p_0, ..., p_{M-1}} be the set of
        occurrences of D_p^{p+l-1}
    let i be the index such that p_i = p
    append ⌊log_2 M⌋ bits of i to K
    append D_p^{p+l-1} c to D
return (D, K)
```

Fig. 2. Recovering redundant bits K in LZ'77. Here, X is the text, K represents the redundant bits, P is the compressed stream of pointers, and D is the decompressed text.

In order to recover additional bits to be used for channel coding, we slightly modify the LZ'77 scheme. The resulting algorithm, called LZS'77, allows one to embed some bits of another binary string K. We define a position i corresponding to the beginning of a phrase to have *multiplicity* M if there exist exactly M matches for the longest prefix that starts at position i in X. The positions with multiplicity $M > 1$ are the places where we can embed some of the bits of K. Specifically, the next $\lfloor \log_2 M \rfloor$ bits will drive the selection of one particular pointer out of the M choices (see Fig. 1). These additional bits can be used for various purposes such as authentication [2] or error correction as described next. In passing, we should acknowledge that the idea of detecting multiple matches of the longest LZ'77 prefix was already considered by Fiala and Greene [6] as a strategy to improve compression. In their scheme C2, the encoder uses a suffix tree to detect two of more copies of the same substring in the database, and only one copy is encoded in the compressed representation.

Suppose again that the initial portion of X, say X_1^{i-1}, has been already parsed. Let $\{(p_0, l, X_{i+l}), (p_1, l, X_{i+l}), \ldots, (p_{M-1}, l, X_{i+l})\}$, $M \ge 1$, be the set of feasible pointers for the longest prefix of X_i^n, where $l > 1$, and $1 \le p_l \le i$ for all $0 \le l \le M-1$. If $M = 1$, we skip to the next phrase, and no extra bits are embedded. When $M > 1$, we use the next $d = \lfloor \log_2 M \rfloor$ bits of K to choose one of the M pointers. Suppose that the first $r-1$ bits of K have already been embedded in previous phrases. We emit the pointer $(p_{K_r^{r+d}}, l, X_{i+l})$, we move the current position to $i+l+1$, and we increment r by d. The complete algorithm is summarized in Fig. 2.

One could extract more bits from the phrase multiplicity by using a start-step-stop binary code [6] that maximizes the code length for a given M. For example, if $M = 6$ one could assign 00 to the first copy, 01 to the second, 100 for the third, 101 for the fourth, 110 for the fifth, and 111 for the sixth. Compared to the original scheme of embedding $\lfloor \log_2 6 \rfloor = 2$ bits by selecting one specific copy out of the first four (among the six available), we would embed an additional bit with probability $2/3$.

We want to stress that these changes do not affect the internal structure of LZ'77 encoding, other than a possible re-shuffling of the pointers. A file compressed with LZS'77 can still be decompressed by a standard LZ'77 algorithm. The fact that LZS'77 is "backward-compatible" makes it possible to deploy it gradually over the existing LZ'77 algorithm, without disrupting service.

From the preceding description, it is clear that the size of the embedded text K depends on the number of longest matches M_n when the first n bits of the input have already been compressed. We analyze M_n for a binary memoryless source, and consider the string $X = X_1X_2X_3\cdots$, where the X_i's are independent and identically distributed (i.i.d.) random variables on the binary alphabet with $\mathsf{Pr}(X_i = 0) = p$ and $\mathsf{Pr}(X_i = 1) = q$. Without loss of generality, we assume throughout the discussion that $q \le p$. Let $X^{(i)}$ denote the ith suffix of X. In other words, $X^{(i)} = X_iX_{i+1}X_{i+2}\cdots$. Consider the *longest* prefix w of $X^{(n+1)}$ such that $X^{(i)}$ also has w as a prefix, for some i with $1 \le i \le n$. Then M_n can be defined as the number of $X^{(i)}$'s (with $1 \le i \le n$) that also have w as a prefix. We formally define the *multiplicity matching parameter* as

$$M_n = \#\{1 \le i \le n \mid X^{(i)} \text{ has } w \text{ as a prefix}\}. \tag{1}$$

Our goal is to understand the probabilistic behavior of the variable M_n. In particular, we compute the jth factorial moment $\boldsymbol{E}[M_n^{\underline{j}}] = \boldsymbol{E}[M_n(M_n-1)\cdots(M_n-j+1)]$, and the limiting distribution $\mathsf{Pr}(M_n = k)$ for large n. We accomplish this by finding the probability generating function $\boldsymbol{E}[u^{M_n}]$ and extracting its asymptotic behavior for large n. The main result presented next is proved in Section III with details explained in Sections IV and V.

Theorem 1: Consider a binary memoryless source, and let $h = -p\log p - q\log q$ be its entropy rate.

(i) There exists $\delta > 0$ depending on p such that the jth factorial moment of M_n is

$$\boldsymbol{E}[M_n^{\underline{j}}] = \Gamma(j)\frac{q(p/q)^j + p(q/p)^j}{h} + \gamma_j(\log_{1/p} n) + O(n^{-\delta}) \tag{2}$$

where Γ is the Euler gamma function, and $\gamma_j(\cdot)$ is a periodic function with mean 0 and small modulus for $\ln p/\ln q$ rational, and asymptotically zero for $\ln p/\ln q$ irrational.

(ii) The probability generating function

$$\boldsymbol{E}[u^{M_n}] = \sum_{k\ge 0}\mathsf{Pr}(M_n = k)u^k$$

is for some $\epsilon > 0$

$$\boldsymbol{E}[u^{M_n}] = -\frac{q\ln(1-pu) + p\ln(1-qu)}{h} + \gamma(\log_{1/p} n, u) + O(n^{-\epsilon}) \tag{3}$$

where $\gamma(\cdot, u)$ is a periodic function with mean 0 and small modulus for $\ln p/\ln q$ rational and asymptotically zero otherwise. More precisely

$$\boldsymbol{E}[u^{M_n}] = \sum_{j=1}^{\infty}\left(\frac{p^jq + q^jp}{jh} - \sum_{k\in\boldsymbol{Z}\setminus\{0\}}\frac{e^{2kr\pi i\log_{1/p} n}\Gamma(z_k)(p^jq+q^jp)(z_k)^{\overline{j}}}{j!(p^{-z_k+1}\ln p + q^{-z_k+1}\ln q)}\right)u^j + O(n^{-\epsilon}) \tag{4}$$

where for $\ln p/\ln q = r/t$ and some $r, t \in \boldsymbol{Z}$ we have $z_k = \frac{2kr\pi i}{\ln p}$. The above translates into

$$\mathsf{Pr}(M_n = j) = \frac{p^jq + q^jp}{jh} - \sum_{k\ne 0}\frac{e^{2kr\pi i\log_{1/p} n}\Gamma(z_k)(p^jq+q^jp)(z_k)^{\overline{j}}}{j!(p^{-z_k+1}\ln p + q^{-z_k+1}\ln q)} + O(n^{-\epsilon}) \tag{5}$$

for some $\epsilon > 0$.

A few remarks are in order. We first comment on the behavior of the function $\gamma_j(t)$. For instance, if we set $p = 1/2$ then

$$|\gamma_j(t)| \le \frac{1}{\ln 2}\sum_{k\ne 0}\left|\Gamma\left(j - \frac{2ki\pi}{\ln 2}\right)\right|.$$

The approximate values of $\frac{1}{\ln 2}\sum_{k\ne 0}\left|\Gamma(j - \frac{2ki\pi}{\ln 2})\right|$ are given in the following table for the first ten values of j.

j	$\frac{1}{\ln 2}\sum_{k\ne 0}\left\|\Gamma\left(j - \frac{2ki\pi}{\ln 2}\right)\right\|$
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}
9	1.7798×10^{3}
10	2.2737×10^{4}

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

For large n we conclude that on average there are $1/h$ eligible pointers and that M_n follows the *logarithmic series distribution*, i.e.,

$$\mathsf{Pr}(M_n = j) \approx \frac{p^jq + q^jp}{jh}$$

plus some small fluctuations. Observe that the probability is maximal for $j = 1$, but $\mathsf{Pr}(M_n = 2)$ is only four times smaller; for $p \gg q$ we also have

$$\mathsf{Pr}(M_n = j+1)/\mathsf{Pr}(M_n = j) \approx pj/(j+1)$$

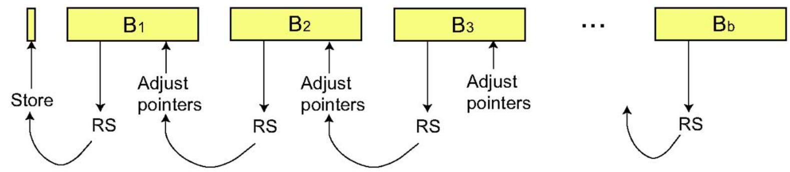

Fig. 3. The right-to-left sequence of operations on the compressed blocks as processed by the LZRS'77 encoder.

thus, the distribution is rather "flat." This bears some immediate consequences for the LZRS'77 scheme since the number of corrected errors depends on $\log M_n$. Knowing that M_n is highly concentrated around its mean is quite reassuring and contributes to a good behavior of the algorithm in practice. In fact, experimental results presented in the next subsection show that there are sufficiently many redundant bits to warrant the use of the LZRS'77 error correction scheme.

B. Error-Resilient LZRS'77 Scheme

We now describe how to use the extra redundant bits to achieve error resilience. Recall that we are protecting the stream of pointers, which is represented by a sequence of bytes. We chose RS codes [19], which are block-based error correcting codes widely used in digital communications and storage.

RS codes belong to the family of Bose–Chaudhuri–Hocquenghem (BCH) codes (see, e.g., [18]). An RS code is specified as $\mathrm{RS}(a, b)$, where a is the size of the block and b is the size of the payload. Let the datum be 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. An RS 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 RS 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 RS codes for $s = 8$ is therefore $\mathrm{RS}(255, 255 - 2e)$. 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 X using the 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 RS parity bits for the block $i+1$ and then it embeds the extra bits in the pointers of block i using the method described in Section II-A. The sequence of operations of the encoder is illustrated in Fig. 3. If one wants to protect the first block as well, then the parity bits of the first block are not embedded, but saved at the beginning of the compressed file. Note that if we decide to store these extra bits at the beginning of the file, the compressed file is not compatible any more with the standard LZ'77 decoder. To keep the file backward compatible one must forgo protecting the first block of the compressed data.

LZRS'77_ENCODER (X, e)
let $b, j, n \leftarrow 1, 1, |X|$
while $j < n$ **do**
  append LZ'77_COMPRESS(X_j) to B_b
  if $|B_b| = 255 - 2e$ **then let** $b \leftarrow b + 1$
for $i \leftarrow b, \ldots, 2$ **do**
  let $RS_i \leftarrow$ REED_SOLOMON_ENCODER(B_i, e)
  embed in the block B_{i-1} the bits RS_i using LZS'77
let $RS_1 \leftarrow$ REED_SOLOMON_ENCODER(B_1, e)
return $RS_1, B_1, B_2, \ldots, B_b$

LZRS'77_DECODER $(RS_1, B_1, B_2, \ldots, B_b, e)$
$D \leftarrow$ empty string
if REED_SOLOMON_DECODER$(B_1 + RS_1, e) =$ errors
  then correct B_1
append LZ'77_DECOMPRESS(B_i) to D
recover RS_2 from the pointers used in B_1 using LZS'77
for $i \leftarrow 2, \ldots, b$ **do**
  if REED_SOLOMON_DECODER$(B_i + RS_i, e) =$ errors
    then correct B_i
  append LZ_DECOMPRESS(B_i) to D
  recover RS_{i+1} from the pointers in B_i using LZS'77
return D

Fig. 4. Error-resilient LZ'77 algorithm. Here X is the text, e is the maximum number of errors that can be corrected in each block of $255 - 2e$ bytes.

If the user selects large values for e, it is possible that the LZ'77 stream may not have enough redundant bits to embed the RS parity bits. This problem can be detected in the encoding phase, when the blocks of size $255 - 2e$ are processed in reverse order. If any block does not have enough redundancy to store the $2e$ extra bytes, an error message is printed, and the user has to choose a smaller value for e.

The decoder receives a sequence of pointers, preceded by the parity bits of the *first block*. It the first breaks, the remainder of the input streams 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 B_1 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 detect and correct 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. A high-level description of the encoder and the decoder is shown in Fig. 4.

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 offline, since it requires the encoder to keep the entire

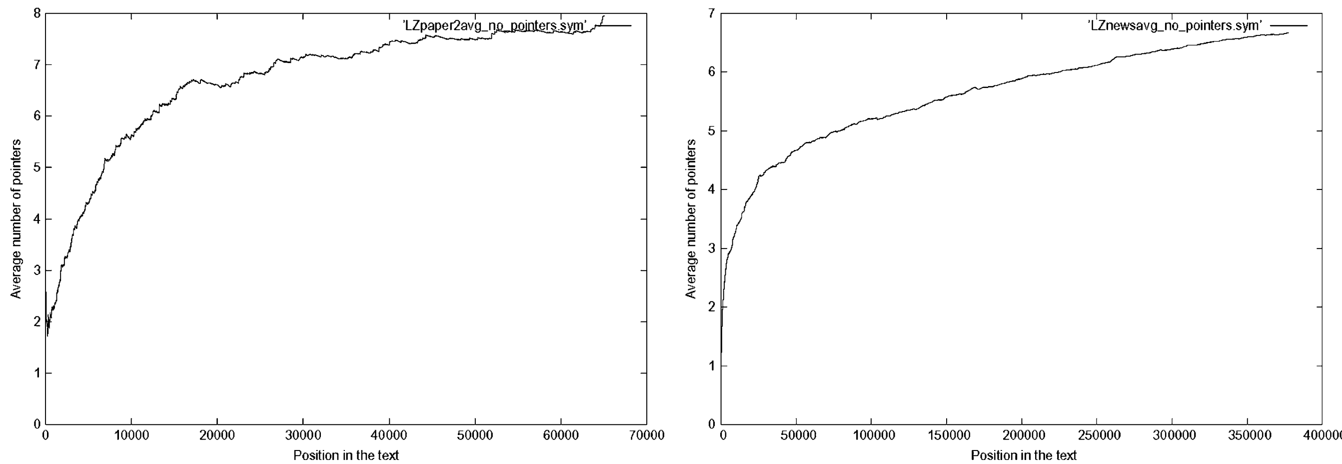

Fig. 5. The average value of the pointer multiplicity M for increasing prefixes of files `paper2` (left), and `news` (right) from the Calgary corpus.

set of buffers in primary memory. The problem can be alleviated by breaking up large inputs in chunks of a size that could be easily stored and processed in main memory.

Even if now the decoder requires two passes, the asymptotic worst case time complexity for the encoder and the decoder is unchanged. If one discounts the extra time spent by error detection/correction algorithm, both encoder and decoder still run in linear time in the size of the input.

C. Experimental Results

In order to validate our theoretical studies presented in Theorem 1 and test the correctness of our LZS'77 scheme, we introduced several implementations. In the first one, we designed an implementation of LZ'77 based on suffix trees [22], and we kept track of the multiplicity M for each phrase of the LZ'77 parsing, when the length of the phrase is greater than two. The average value of M is shown in Fig. 5, for increasing lengths of the prefixes. Note that for both graphs, the average for M appears to converge asymptotically to a constant, as Theorem 1 suggests.

In the second, we modified the code of `gzip-1.2.4` to evaluate the impact of our method on compression performance. The tool `gzip` is an implementation of the *sliding-window* variant of LZ'77, that issues pointers in a fixed-size window preceding the current position. Among the various parameters available, `gzip` allows the user to specify the level of compression from level `-1` (worst, fastest) to level `-9` (best, slowest). This parameter mainly controls the size of the sliding window (bigger windows correspond to higher compression but slower programs), but also activates the "lazy evaluation" (or "nongreedy parsing") strategy [9]. The lazy evaluation scheme is active from level `-4` to level `-9`.

The modified `gzip`, called `gzipS`, directly implements LZS'77 as described in Section II-A. It allows the user to specify a second file, which contains the text to be embedded in the pointers. The compression performance of the `gzipS` with respect to the original `gzip` was measured, and it is illustrated in Table I on the Calgary corpus dataset. Since a nongreedy parsing would introduce additional complexity in the LZS'77 decoder to recover correctly the extra redundant

TABLE I
THE COMPRESSION OF "GZIP -3" VERSUS "GZIPS -3" FOR THE FILES OF THE CALGARY CORPUS; THE LAST COLUMN SHOWS THE TOTAL NUMBER OF AVAILABLE BYTES FOR ERROR CORRECTION

file size	`gzip`	`gzipS`	*file*	*redundant*
111,261	39,473	39,511	`bib`	1,721
768,771	333,776	336,256	`book1`	14,524
610,856	228,321	228,242	`book2`	10,361
102,400	69,478	71,168	`geo`	4,101
377,109	155,290	156,150	`news`	5,956
21,504	10,584	10,783	`obj1`	353
246,814	89,467	89,757	`obj2`	3,628
53,161	20,110	20,204	`paper1`	937
82,199	32,529	32,507	`paper2`	1,551
46,526	19,450	19,567	`paper3`	893
13,286	5,853	5,898	`paper4`	249
11,954	5,252	5,294	`paper5`	210
38,105	14,433	14,506	`paper6`	738
513,216	62,357	61,259	`pic`	3,025
39,611	14,510	14,660	`progc`	736
71,646	18,310	18,407	`progl`	1,106
49,379	12,532	12,572	`progp`	741
93,695	22,178	22,098	`trans`	1,201

bits, we used the compression level `-3` but we increased the size of the sliding window to the one used in level `-9` in order to maximize the chances to find multiple copies.

According to the documentation, in the presence of multiple copies of the longest prefix `gzip` always chooses the most recent occurrence in the sliding window. Pointers are represented as a pair *(displacement, length)* where the displacement is the distance between the copy in the database and the current position, and they are Huffman encoded. By choosing always the most recent occurrence `gzip` produces frequent short displacements that get shorter representations in the Huffman tree. Because of this, the embedding of the message slightly degrades the compression performance, on the order of 1%–2% on average for the files in the Calgary corpus. A file compressed with `gzipS` can be still be decompressed by the original `gzip`, and therefore is backward compatible.

Finally, in the last implementation, we coded the error-resilient LZRS'77. The prototype implementation is written in Python, with calls to C public-domain code that implements

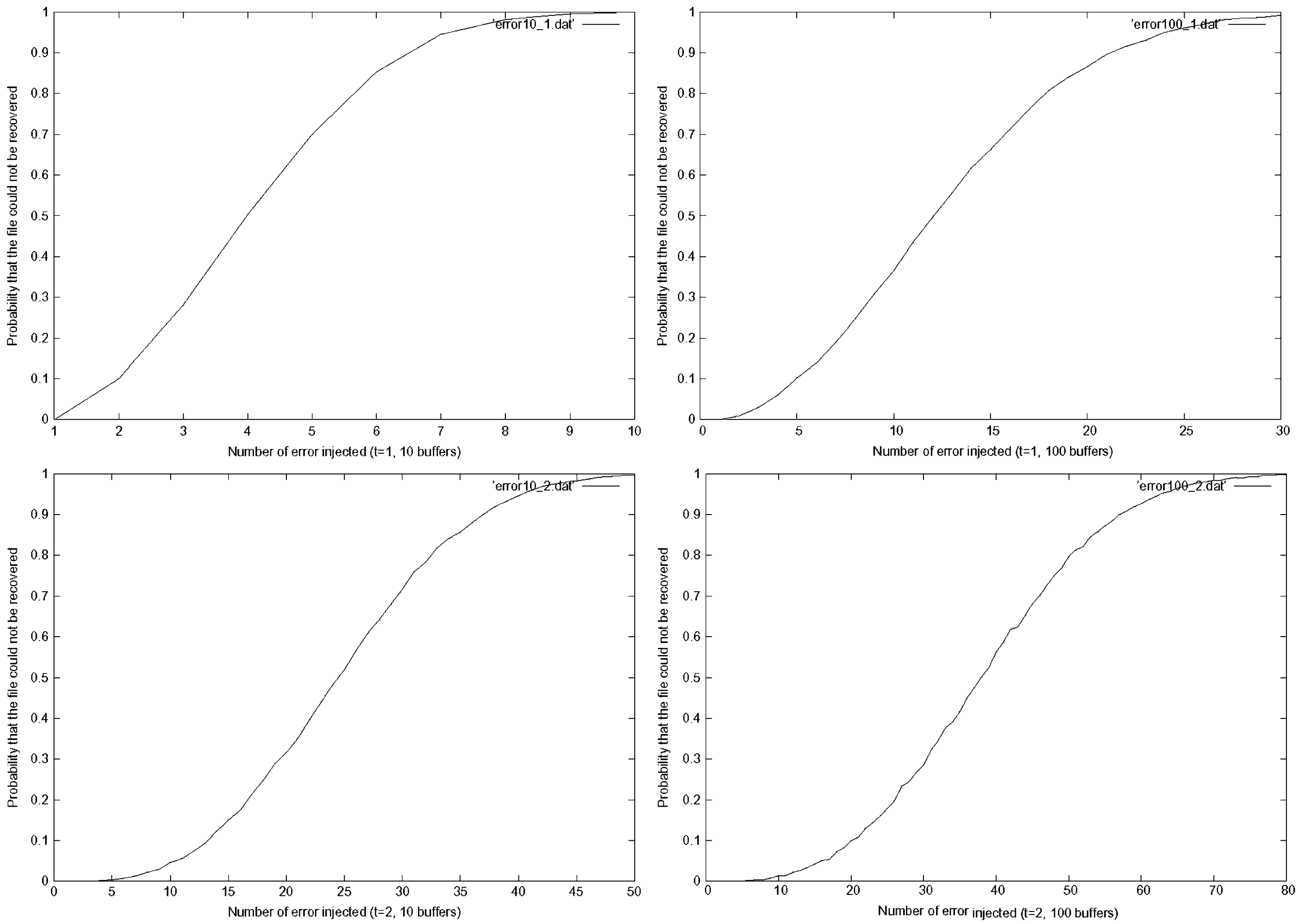

Fig. 6. The probability that a file of b blocks could not be recovered correctly, for increasing number of errors uniformly distributed over the blocks. Top-left: $e = 1$ and $b = 10$, top-right: $e = 1$ and $b = 100$, lower-left: $e = 2$ and $b = 10$, lower-right: $e = 2$ and $b = 100$.

the RS encoder/decoder [14]. Based on the considerations mentioned in the Introduction, we initially choose $e = 1$ and $e = 2$ which require, respectively, at least two and four parity bytes on a block of data of size $255 - 2e$. We experimented with the resilience to errors by introducing a controlled number of errors uniformly distributed over the b blocks of the compressed file. The graphs in Fig. 6 show the probability that the file did *not* uncompress correctly for increasing numbers of errors for different choices of e and b.

For example, using $e = 2$ over 100 blocks, LZRS'77 is able to decompress the file correctly with 20 uniformly distributed errors, 90% of the time. In this case, the compressed file size would be about 25 500 bytes. Assuming that LZRS'77 loses 1%–2% on average in compression performance compared to LZ'77, we could conclude that we could save 255–510 bytes by using the original LZ'77. The savings should be compared to the 400 parity bytes that are embedded in the LZRS'77 file.

III. Streamlined Analysis

In this section, we guide the reader through the main ideas of the proof of Theorem 1 with details explained in the last two sections.

We recall the definition of the multiplicity matching parameter. The variable M_n represents the number of longest matches within the first n symbols of the database as formally expressed in (1). We now provide an alternative definition of M_n via *suffix trees*. A suffix tree is a trie built from suffixes of a single string. A *trie* is a digital tree built over, say n, strings (the reader is referred to [15], [24], [27] for an in-depth discussion of digital trees). A string is stored in an external node of a trie; the path length to such a node is the shortest prefix of the string that is not a prefix of any other strings (cf. Fig. 7). For a binary alphabet, each branching node in a trie is a binary node. A special case of a trie structure is a *suffix trie* (tree) which is a trie built over suffixes of a *single* string.

Now we can redefine M_n via suffix trees. First, build a suffix tree from the first $n + 1$ suffixes of X. Consider the *insertion point* of the $(n + 1)$th suffix. Then M_n is exactly equal to the *number of leaves* in the subtree rooted at the branching point of the $(n + 1)$th insertion. For instance, suppose that the $(n + 1)$th suffix starts with $w\beta$ for some $\beta \in \mathcal{A} := \{0, 1\}$, and some $w \in \mathcal{A}^*$. Then, examining the first n suffixes, if there are exactly k suffixes that begin with $w\alpha$ (where $\alpha = 1 \oplus \beta$ where $\oplus$ is addition modulo 2), and the other $n - k$ suffixes do

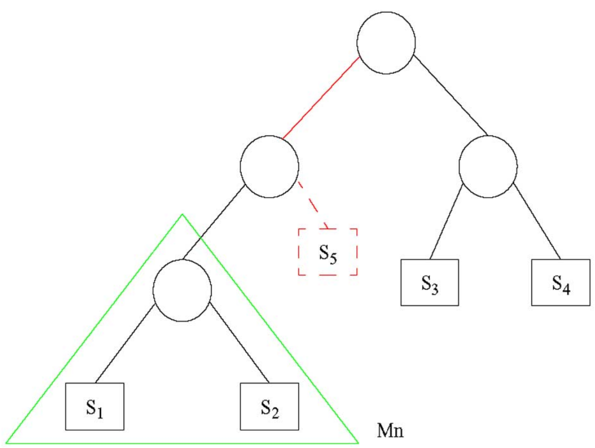

Fig. 7. A trie and its multiple matching parameter M_4 after inserting string S_5.

not begin with w, we conclude that $M_n = k$. Fig. 7 illustrates this scenario.

Our goal is to study M_n in a suffix tree built from a string X generated by a binary memoryless source. Unfortunately, the strings in a suffix tree are highly dependent on each other; thus, a precise analysis of M_n is quite difficult. For this reason, we first analyze the analogous situation in a trie built over *independent* strings. Specifically, in Section IV we analyze the distribution and moments of a random variable with similar properties, namely, M_n^I, via the analysis of *independent tries*, using analytical poissonization and depoissonization, the Mellin transform, and complex analysis (cf. [27]). To define M_n^I, we consider the situation described above, but we build a trie from $n+1$ *independent* strings from $\mathcal{A}^*$. So we consider independent $X(i)$'s (more specifically, $X(i) = X_1(i)X_2(i)X_3(i)\cdots$, where the $X_j(i)$'s are i.i.d. random variables). We let w denote the *longest* prefix of $X(n+1)$ such that $X(i)$ also has w as a prefix, for some i with $1 \le i \le n$. Then M_n^I is defined as the number of $X(i)$'s (with $1 \le i \le n$) that also have w as a prefix, that is,

$$M_n^I = \#\{1 \le i \le n \mid X(i) \text{ has } w \text{ as a prefix}\}. \tag{6}$$

In order to analyze M_n^I, we define the alignment $C_{j_1,\ldots,j_k}$ among k strings $X(j_1),\ldots,X(j_k)$ as the length of the longest common prefix of the k strings. The kth 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 kth string. Note $D_{n+1}(n+1) = \max_{1\le j\le n} C_{j,n+1} + 1$. Thus, in the context of tries

$$M_n^I = \#\{j \mid 1 \le j \le n, C_{j,n+1} + 1 = D_{n+1}(n+1)\}.$$

That is, M_n^I is the size of a subtree rooted at the branching point of a new insertion. We analyze M_n^I through generating functions. Define the exponential generating functions

$$G(z,u) = \sum_{n\ge 0} \boldsymbol{E}[u^{M_n^I}]\frac{z^n}{n!},$$
$$F_j(z) = \sum_{n\ge 0} \boldsymbol{E}[(M_n^I)^{\underline{j}}]\frac{z^n}{n!}$$

for complex $u \in \boldsymbol{C}$ and $j \in \boldsymbol{N}$. A simple combinatorial argument, based on our discussion above, shows that

$$\Pr\{M_n^I = k\} = \sum_{\substack{w\in\mathcal{A}^*\\ \alpha\in\mathcal{A}}} \Pr(w\beta)\binom{n}{k}\Pr(w\alpha)^k(1-\Pr(w))^{n-k}. \tag{7}$$

It follows that

$$G(z,u) = 1 + \sum_{\substack{w\in\mathcal{A}^*\\ \alpha\in\mathcal{A}}} \Pr(w\beta)\left(e^{z(1-\Pr(w)+u\Pr(w\alpha))} - e^{z(1-\Pr(w))}\right)$$
$$F_j(z) = \sum_{\substack{w\in\mathcal{A}^*\\ \alpha\in\mathcal{A}}} \Pr(w\beta)e^{z(1-\Pr(w\beta))}(\Pr(w\alpha)z)^j.$$

We derive in Section IV asymptotics using poissonization, the Mellin transform, and depoissonization; details are given in the next section. *These methods allow us to establish Theorem 1 with M_n replaced by M_n^I.*

Once we have established the probabilistic properties of M_n^I, we can deal with the more difficult problem, namely, the multiplicity matching parameter M_n in a suffix tree. We show that M_n has a similar asymptotic distribution as M_n^I. To prove this, we compare the distribution of M_n in suffix trees versus the distribution of M_n^I in independent tries. Specifically, we prove the following theorem.

Theorem 2: There exists $\epsilon > 0$ such that, for some $\delta > 0$ and for all $|u| < 1 + \delta$

$$|M_n(u) - M_n^I(u)| = O(n^{-\epsilon}). \tag{8}$$

As a consequence, there exists $b > 1$ such that

$$\Pr(M_n = k) - \Pr(M_n^I = k) = O(n^{-\epsilon}b^{-k}) \tag{9}$$

for large n.

A detailed analysis of M_n is presented in Section V. Briefly, our proof technique follows these lines. We let

$$M(z,u) = \sum_{1\le k,n\le\infty} \Pr(M_n = k)u^k z^n$$
$$M^I(z,u) = \sum_{1\le k,n\le\infty} \Pr(M_n^I = k)u^k z^n$$

denote the bivariate generating functions for M_n and M_n^I, respectively. To study these generating functions, we consider the w's defined above. Specifically, for $M(z,u)$, we recall from (1) that if w denotes the longest prefix of $X^{(n+1)} = X_{n+1}X_{n+2}X_{n+3}\cdots$ that appears as a prefix of any $X^{(i)} = X_iX_{i+1}X_{i+2}\cdots$, then M_n enumerates the number of such occurrences of w. This approach to $M(z,u)$ allows us to sum over all $w \in \mathcal{A}^*$ instead of summing over $k,n \in \boldsymbol{N}$. Similarly, for $M^I(z,u)$, we utilize (6) to determine that if w denotes the longest prefix of $X^{(n+1)} = X_1(n+1)X_2(n+1)X_3(n+1)\cdots$ that appears as a prefix of any $X_1(i)X_2(i)X_3(i)\cdots$, then M_n^I is precisely the number of such occurrences of w. Therefore, to evaluate $M^I(z,u)$, we can sum over all $w \in \mathcal{A}^*$ instead of summing over the integers k and n.

We note that the $X^{(i)}$'s in a suffix tree are highly dependent on each other. In fact, if $i \geq j$, then $X^{(i)} = X_i X_{i+1} X_{i+2} \cdots$ is a substring of $X^{(j)} = X_j X_{j+1} X_{j+2} \cdots$. This dependency makes the derivation of the bivariate generating function $M(z,u)$ quite difficult. We overcome this hurdle by succinctly describing the degree to which a suffix of X can overlap with itself. We accomplish this by utilizing the autocorrelation polynomial $S_w(z)$ of a word w, which measures the amount of overlap of a word w with itself. The autocorrelation polynomial is defined as (cf. [10], [17], [20])

$$S_w(z) = \sum_{k \in \mathcal{P}(w)} \mathsf{Pr}(w_{k+1}^m) z^{m-k} \tag{10}$$

where $\mathcal{P}(w)$ denotes the set of positions k of w satisfying $w_1 \dots w_k = w_{m-k+1} \cdots w_m$, that is, w's prefix of length k is equal to w's suffix of length k. Via the autocorrelation polynomial, we are able to surmount the difficulties inherent in the overlapping suffixes. Thus, using $S_w(z)$, we obtain a succinct description of the bivariate generating function $M(z,u)$. The autocorrelation polynomial is well understood; we utilize several results about $S_w(z)$ from [17] and [20]. In particular, when comparing $M(z,u)$ and $M^I(z,u)$, it is extremely useful to note that the autocorrelation polynomial $S_w(z)$ is close to 1 with high probability (for $|w|$ large), that is, for a random string w there is not much overlap.

In order to obtain information about the difference of the above two random variables, we analyze $Q(z,u) = M(z,u) - M^I(z,u)$ using residue analysis. We make a comparison of the poles of $M(z,u)$ and $M^I(z,u)$ using Cauchy's theorem (integrating with respect to z). As a result, we prove that $Q_n(u) := [z^n]Q(z,u) = O(n^{-\epsilon})$ uniformly for $|u| \leq p^{-1/2}$ as $n \to \infty$. Then we use another application of Cauchy's theorem (integrating with respect to u). Specifically, we extract the coefficient

$$\mathsf{Pr}(M_n = k) - \mathsf{Pr}(M_n^I = k) = [u^k z^n] Q(z,u).$$

This establishes Theorem 2.

IV. Analysis of Independent Tries

In this section, we prove Theorem 1 for M_n^I instead of M_n. Our first step is poissonization. Then we utilize the Mellin transform and complex analysis; thus, we obtain asymptotic descriptions of the distribution and factorial moments of M_n^I. Since these results are valid for the *poissonized* model of the problem, we must depoissonize our results in order to find the asymptotic distribution and factorial moments of M_n^I in the original model.

A. Poissonization

We first utilize analytical poissonization. The idea is to replace the fixed-size population model by a poissonized model in which the number of strings is a Poisson random variable with mean n. We apply the Poisson transform to the exponential generating functions $G(z,u)$ and $F_j(z)$, which yields

$$\tilde{G}(z,u) = \sum_{n \geq 0} \boldsymbol{E}[u^{M_n^I}] \frac{z^n}{n!} e^{-z}$$
$$\tilde{F}_j(z) = \sum_{n \geq 0} \boldsymbol{E}\left[(M_n^I)^{\underline{j}}\right] \frac{z^n}{n!} e^{-z}. \tag{11}$$

We observe that

$$\tilde{G}(z,u) = e^{-z} + \sum_{\substack{w \in \mathcal{A}^* \\ \alpha \in \mathcal{A}}} \mathsf{Pr}(w\beta) \left(e^{-z\mathsf{Pr}(w)(1-u\mathsf{Pr}(\alpha))} - e^{-z\mathsf{Pr}(w)}\right)$$
$$\tilde{F}_j(z) = \sum_{\substack{w \in \mathcal{A}^* \\ \alpha \in \mathcal{A}}} \mathsf{Pr}(w\beta) e^{-z\mathsf{Pr}(w\beta)} (\mathsf{Pr}(w\alpha)z)^j$$

by applying (7) to (11).

B. Mellin Transform

If f is a complex-valued function which is continuous on $(0,\infty)$ and is locally integrable, then the Mellin transform of f is defined as

$$\mathcal{M}[f(x);s] = f^*(s) = \int_0^\infty f(x) x^{s-1}\, dx$$

(see [8] and [27]).

We define $\hat{G}(x,u) = \tilde{G}(x,u) - 1$ (so that $\hat{G}(x,u) = O(x)$ as $x \to 0$). If $u \in \boldsymbol{R}$ with $u < \min\{1/p, 1/q\}$ and if $\Re(s) \in (-1,0)$, then

$$\hat{G}^*(s,u) = \Gamma(s) \frac{q(1-pu)^{-s} + p(1-qu)^{-s} - p^{-s+1} - q^{-s+1}}{1 - p^{-s+1} - q^{-s+1}}.$$

If $j \in \boldsymbol{N}$ and $\Re(s) \in (-j,0)$, then

$$\tilde{F}_j^*(s) = \Gamma(s+j) \frac{p^j q^{-s-j+1} + q^j p^{-s-j+1}}{1 - p^{-s+1} - q^{-s+1}}.$$

We next invert the Mellin transform, computing

$$\tilde{F}_j(x) = \frac{1}{2\pi i} \int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty} \tilde{F}_j^*(s) x^{-s}\, ds$$
$$\hat{G}(x,u) = \frac{1}{2\pi i} \int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty} \hat{G}^*(s,u) x^{-s}\, ds$$

since $c = -1/2$ is in the fundamental strip of $\hat{G}(x,u)$.

C. Results for the Poisson Model

We restrict our attention to the case where $\ln p / \ln q$ is rational. Thus, we can write $\ln p / \ln q = r/t$ for some relatively prime $r, t \in \boldsymbol{Z}$. Then, by a theorem of Jacquet and Schachinger (see [27]), we know that the set of poles of $\tilde{F}_j^*(s)x^{-s}$ is exactly $\{z_k = \frac{2kr\pi i}{\ln p} \mid k \in \boldsymbol{Z}\}$. We also observe that $\tilde{F}_j^*(s)x^{-s}$ has simple poles at each z_k. Now we assume that $u \neq 1$. Then $\hat{G}^*(s,u)x^{-s}$ has the same set of poles as $\tilde{F}_j^*(s)x^{-s}$, each of which is a simple pole.

Using the Cauchy residue theorem [1], if $j \in \boldsymbol{N}$ and $z_k = \frac{2kr\pi i}{\ln p}$, then

$$\tilde{F}_j(x) = \sum_{k \in \boldsymbol{Z}} -\mathrm{Res}[\tilde{F}_j^*(s)x^{-s};\ z_k] + O(x^{-L})$$

and

$$\hat{G}(x,u) = \sum_{k \in \boldsymbol{Z}} -\mathrm{Res}[\hat{G}^*(s,u)x^{-s};\ z_k] + O(x^{-L}).$$

It follows that, for $j \in \boldsymbol{N}$

$$\tilde{F}_j(x) = \Gamma(j)\frac{q(p/q)^j + p(q/p)^j}{h} + \delta_j(\log_{1/p} x) + O(x^{-L})$$

where $h = -p\ln p - q\ln q$ denotes the entropy and

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

Also

$$\hat{G}(x,u) = -\frac{q\ln(1-pu) + p\ln(1-qu)}{h} - 1 + \gamma(\log_{1/p} x, u) + O(x^{-L}) \quad (12)$$

where

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

As an immediate corollary of (12), we see that

$$\tilde{G}(x,u) = -\frac{q\ln(1-pu) + p\ln(1-qu)}{h} + \gamma(\log_{1/p} x, u) + O(x^{-L}).$$

We note that, if $\ln p/\ln q$ is irrational and u is fixed, then $\gamma_j(x) \to 0$ and $\gamma(x,u) \to 0$ as $x \to \infty$. Thus, γ_j and $\gamma(\cdot, u)$ do not exhibit fluctuation when $\ln p/\ln q$ is irrational.

D. Depoissonization

Recall that in the original problem statement n is a large, fixed integer. Most of our analysis has utilized a model where n is a Poisson random variable. Therefore, to obtain results about the problem we originally stated, it is necessary to depoissonize our results.

Using depoissonization results of [12] and [27], we can depoissonize our results (cf. [28], [29]). Our conclusion is that Theorem 1 holds if we replace M_n by M_n^I.

V. Analysis of LZS'77 Via Suffix Trees

In this section, we establish Theorem 2, and as a consequence, we immediately prove the validity of our main result (namely, Theorem 1) for M_n.

Consider a suffix tree built from n suffixes of $X = X_1X_2X_3\cdots$, where the X_i's are i.i.d. random variables on the alphabet $\mathcal{A} = \{0,1\}$ with $\Pr(X_i = 0) = p$ and $\Pr(X_i = 1) = q$. As before, without loss of generality, $q \le p$. Let $X^{(i)}$ denote the ith suffix of X. Then M_n is defined as the number of $X^{(i)}$'s (with $1 \le i \le n$) that also have w as a prefix, that is,

$$M_n = \#\{1 \le i \le n \mid X^{(i)} \text{ has } w \text{ as a prefix}\}.$$

In Section III, we redefined M_n as the multiplicity matching parameter in a suffix tree built over X (cf. Fig. 7). In this section, we analyze M_n and compare its distribution to that of M_n^I. In short, we first obtain the bivariate generating functions for M_n and M_n^I, denoted as $M(z,u)$ and $M^I(z,u)$, respectively. (In particular, we rederive $M^I(z,u)$ in such a way that a comparison to $M(z,u)$ is very natural.) Next, we prove that $M(z,u)$ can be analytically continued from the unit disk to a larger disk. Afterward, we determine the poles of $M(z,u)$ and $M^I(z,u)$. We write $Q(z,u) = M(z,u) - M^I(z,u)$; we use Cauchy's theorem to prove that $Q_n(u) := [z^n]Q(z,u) \to 0$ uniformly for $u \le p^{-1/2}$ as $n \to \infty$. Then we apply Cauchy's theorem again to prove that

$$\Pr(M_n = k) - \Pr(M_n^I = k) = [u^k z^n]Q(z,u) = O(n^{-\epsilon}b^{-k})$$

for some $\epsilon > 0$ and $b > 1$.

We conclude that the distribution of the multiplicity matching parameter M_n is asymptotically the same in suffix trees as in tries built over independent strings, proving Theorem 2, i.e., M_n and M_n^I have asymptotically the same distribution. Therefore, M_n also follows the logarithmic series distribution plus some fluctuations, as claimed by Theorem 1.

A. Multiplicity Matching Parameter of Independent Tries

First we rederive the bivariate generating function for M_n^I using a different approach (the so-called "string-ruler" method) that is well suited for suffix trees. We deal here with a trie built over the *independent* strings $X^{(1)}, \dots, X^{(n+1)}$, where $X^{(i)} = X_1(i)X_2(i)X_3(i)\dots$ and $\{X_j(i) \mid i,j \in \boldsymbol{N}\}$ is a collection of i.i.d. random variables with $\Pr(X_j(i) = 0) = p$ and $\Pr(X_j(i) = 1) = q = 1 - p$. We let w denote the *longest prefix* of both $X^{(n+1)}$ and at least one other string $X^{(i)}$ for some $1 \le i \le n$. We write β to denote the $(|w|+1)$th character of $X^{(n+1)}$. When $M_n^I = k$, we conclude that exactly k strings $X^{(i)}$ have $w\alpha$ as a prefix, and the other $n-k$ strings $X^{(i)}$ do not have w as a prefix at all. Thus, the generating function for M_n^I is exactly

$$\begin{aligned} M^I(z,u) &= \sum_{n=1}^{\infty}\sum_{k=1}^{\infty} \Pr(M_n^I = k)u^k z^n \\ &= \sum_{n=1}^{\infty}\sum_{k=1}^{\infty}\sum_{\substack{w\in\mathcal{A}^*\\ \alpha\in\mathcal{A}}} \Pr(w\beta)\binom{n}{k}\Pr(w\alpha)^k \\ &\quad \times (1-\Pr(w))^{n-k}u^k z^n. \end{aligned}$$

After simplifying, it follows immediately that

$$M^I(z,u) = \sum_{\substack{w\in\mathcal{A}^*\\ \alpha\in\mathcal{A}}} \frac{u\Pr(\beta)\Pr(w)}{1 - z(1-\Pr(w))} \times \frac{z\Pr(w)\Pr(\alpha)}{1 - z(1 + u\Pr(w)\Pr(\alpha) - \Pr(w))}. \quad (13)$$

The same line of reasoning about $M^I(z,u)$ can be applied in the next subsection to derive the generating function $M(z,u)$ for M_n, but the situation will be more complicated because the occurrences of w can overlap.

B. Multiplicity Matching Parameter of Suffix Trees

Now we obtain the bivariate generating function for M_n, which is the multiplicity matching parameter for a suffix tree built over the first $n+1$ *suffixes* $X^{(1)}, \dots, X^{(n+1)}$ of a string X (i.e., $X^{(i)} = X_iX_{i+1}X_{i+2}\dots$). The bivariate generating function for the multiplicity matching parameter is much more difficult to derive in the dependent (suffix tree) case than in the independent (trie) case, because the suffixes of X are dependent on each other. We let w denote the *longest prefix* of both $X^{(n+1)}$ and at least one $X^{(i)}$ for some $1 \le i \le n$. We write β to denote the $(|w|+1)$th character of $X^{(n+1)}$; when $M_n = k$, we conclude that exactly k suffixes $X^{(i)}$ have $w\alpha$ as a prefix, and the other $n-k$ strings $X^{(i)}$ do not have w as a prefix at all. Thus, we are interested in finding strings with exactly k occurrences of $w\alpha$, ended on the right by an occurrence of $w\beta$, with no other occurrences of w at all. This set of words constitutes the language $\mathcal{R}_w \alpha (\mathcal{T}_w^{(\alpha)}\alpha)^{k-1}\mathcal{T}_w^{(\alpha)}\beta$, where

$$\begin{aligned}\mathcal{R}_w &= \{v \in \mathcal{A}^* \mid v \text{ contains exactly one occurrence} \\ &\qquad\qquad \text{of } w, \text{located at the right end}\} \\ \mathcal{T}_w^{(\alpha)} &= \{v \in \mathcal{A}^* \mid w\alpha v \text{ contains exactly two occurrences} \\ &\qquad\qquad \text{of } w, \text{located at the left and right ends}\}.\end{aligned}$$

Thus, the generating function for M_n is

$$\begin{aligned}M(z,u) = \sum_{k=1}^{\infty} \sum_{\substack{w\in\mathcal{A}^* \\ \alpha\in\mathcal{A}}} \sum_{s\in\mathcal{R}_w} &\mathsf{Pr}(s\alpha) z^{|s|+1} u \\ &\times \left(\sum_{t\in\mathcal{T}_w^{(\alpha)}} \mathsf{Pr}(t\alpha) z^{|t|+1} u\right)^{k-1} \\ &\times \sum_{v\in\mathcal{T}_w^{(\alpha)}} \mathsf{Pr}(v\beta) z^{|v|+1-|w|-1}. \qquad (14)\end{aligned}$$

Using combinatorics on words, as discussed in [10], [17], [20], and as applied in [28], we derive a form of $M(z,u)$ that we summarize as follows.

Theorem 3: Let $M(z,u) := \sum_{n=1}^{\infty}\sum_{k=1}^{\infty} \mathsf{Pr}(M_n = k) u^k z^n$ denote the bivariate generating function for M_n, the multiplicity matching parameter of a suffix tree built over the first $n+1$ *suffixes* $X^{(1)},\ldots,X^{(n+1)}$ of a string X. Then

$$\begin{aligned}M(z,u) = \sum_{\substack{w\in\mathcal{A}^* \\ \alpha\in\mathcal{A}}} &\frac{u\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{D_w(z)} \\ &\times \frac{D_{w\alpha}(z) - (1-z)}{D_w(z) - u(D_{w\alpha}(z) - (1-z))} \qquad (15)\end{aligned}$$

for $|u| < 1$ and $|z| < 1$. Here

$$D_w(z) = (1-z)S_w(z) + z^{|w|}\mathsf{Pr}(w)$$

and $S_w(z)$ denotes the autocorrelation polynomial for w, defined in (10).

Proof: The generating functions associated with $\mathcal{R}_w$ and $\mathcal{T}_w^{(\alpha)}$ are, respectively

$$R_w(z) := \sum_{v\in\mathcal{R}_w} \mathsf{Pr}(v) z^{|v|}$$

and

$$T_w^{(\alpha)}(z) := \sum_{v\in\mathcal{T}_w^{(\alpha)}} \mathsf{Pr}(v) z^{|v|}.$$

From [20], we know $R_w(z)/z^{|w|} = \mathsf{Pr}(w)/D_w(z)$, so we simplify (14) to obtain

$$M(z,u) = \sum_{\substack{w\in\mathcal{A}^* \\ \alpha\in\mathcal{A}}} \frac{u\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{D_w(z)} \frac{\mathsf{Pr}(\alpha) z T_w^{(\alpha)}(z)}{1 - \mathsf{Pr}(\alpha) z u T_w^{(\alpha)}(z)}. \qquad (16)$$

To obtain an explicit form of $T_w^{(\alpha)}(z)$, we define

$$\begin{aligned}\mathcal{M}_w := \{v \in \mathcal{A}^* \mid wv &\text{contains exactly two occurrences} \\ &\text{of } w, \text{ located at the left and right ends}\}\end{aligned}$$

and

$$\mathcal{H}_w^{(\alpha)} := \mathcal{M}_w \cap (\alpha\mathcal{A}^*).$$

We observe that $\alpha\mathcal{T}_w^{(\alpha)} = \mathcal{H}_w^{(\alpha)}$. Thus, (16) simplifies to

$$M(z,u) = \sum_{\substack{w\in\mathcal{A}^* \\ \alpha\in\mathcal{A}}} \frac{u\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{D_w(z)} \frac{H_w^{(\alpha)}(z)}{1 - uH_w^{(\alpha)}(z)}. \qquad (17)$$

So we can complete the proof of Theorem 3 by establishing Lemma 1 below. □

Lemma 1: Let $\mathcal{H}_w^{(\alpha)}$ denote the subset of words from $\mathcal{M}_w$ that begin with α. The generating function

$$H_w^{(\alpha)}(z) = \sum_{v\in\mathcal{H}_w^{(\alpha)}} \mathsf{Pr}(v) z^{|v|}$$

is

$$H_w^{(\alpha)} = \frac{D_{w\alpha}(z) - (1-z)}{D_w(z)}$$

where $D_w(z) = (1-z)S_w(z) + z^{|w|}\mathsf{Pr}(w)$.

Proof: We utilize a method relying on combinatorics of correlation with borders, as discussed in [20].

We define $\mathcal{H} = \{w\alpha, w\beta\}$; also let $H_1 = w\alpha$ and $H_2 = w\beta$. We write

$$\mathbb{H} = \begin{bmatrix} \mathsf{Pr}(H_1) & \mathsf{Pr}(H_1) \\ \mathsf{Pr}(H_2) & \mathsf{Pr}(H_2) \end{bmatrix}.$$

We define $\mathcal{A}_{H,F} = \{F_{k+1}^m \mid H_{m-k+1}^m = F_1^k\}$ as a generalization of the autocorrelation polynomial, describing the overlap of H with F. This yields

$$\begin{aligned}A_{w\alpha,w\alpha}(z) &= S_{w\alpha}(z) \\ A_{w\alpha,w\beta}(z) &= (S_{w\alpha}(z) - 1)\mathsf{Pr}(\beta)/\mathsf{Pr}(\alpha) \\ A_{w\beta,w\alpha}(z) &= (S_{w\beta}(z) - 1)\mathsf{Pr}(\alpha)/\mathsf{Pr}(\beta) \\ A_{w\beta,w\beta}(z) &= S_{w\beta}(z).\end{aligned}$$

Next we define $\mathbb{D}(z) = (1-z)\mathbb{A}(z) + z^{m+1}\mathbb{H}^T$, where $\mathbb{H}^T$ denotes the transpose of $\mathbb{H}$, and where

$$\mathbb{A}(z) := \begin{bmatrix} A_{w\alpha,w\alpha}(z) & A_{w\alpha,w\beta}(z) \\ A_{w\beta,w\alpha}(z) & A_{w\beta,w\beta}(z) \end{bmatrix}.$$

We also define $\mathbb{M}(z) = (\mathbb{D}(z) + (z-1)\mathbb{I})\mathbb{D}(z)^{-1}$, where $\mathbb{I}$ denotes the 2×2 identity matrix. Then

$$\begin{aligned}&\mathbb{M}_{1,2}(z) \\ &= \frac{(1-z)(S_{w\alpha}(z) - 1)\mathsf{Pr}(\beta)/\mathsf{Pr}(\alpha) + z^{m+1}\mathsf{Pr}(w\beta)}{(1-z)S_w(z) + z^m\mathsf{Pr}(w)}. \qquad (18)\end{aligned}$$

We know by [20] that the set enumerated by $\mathbb{M}_{1,2}(z)$, namely, $\mathbb{M}_{1,2}$, is exactly the set of words v such that $w\alpha v$ has exactly one occurrence of $w\alpha$ and one occurrence of $w\beta$, at the left and right ends, respectively. If we write $u\beta = v$ (for the appropriate $u \in \mathcal{A}^*$), this happens if and only if $w\alpha u$ has exactly two occurrences of w, at the left and right ends. Therefore, $\mathbb{M}_{1,2} = \mathcal{T}_w^{(\alpha)} \cdot \beta$. By also recalling $\mathcal{H}_w^{(\alpha)} = \alpha \mathcal{T}_w^{(\alpha)}$, we can easily simplify (18), thereby completing the proof of the lemma. □

Lemma 1 was the last required ingredient in the proof of Theorem 3.

C. Analytic Continuation

In order to establish (9) of Theorem 2 we need to first note that $M(z,u)$ can be analytically continued.

Theorem 4: The generating function $M(z,u)$ can be analytically continued for $|u| \leq \delta^{-1}$ and $|z| < 1$.

The proof requires several lemmas and observations, all found in [28]. We merely state the main lemma underlying this theorem.

Lemma 2: If $0 < r < 1$, then there exists $C > 0$ (depending on r) such that

$$|D_w(z) - u(D_{w\alpha}(z) - (1-z))| \geq C$$

for $|z| \leq r$ (and, as before, $|u| \leq \delta^{-1}$).

D. Singularity Analysis

We need some auxiliary results before we prove our main result of this section, namely Theorem 2. We first determine (for $|u| \leq \delta^{-1}$) the zeroes of $D_w(z) - u(D_{w\alpha}(z) - (1-z))$ and in particular the zeroes of $D_w(z)$.

For instance, we state without proof, the following lemma. (See [28] for a rigorous proof.)

Lemma 3: There exists an integer $K_2 \geq 1$ such that, for u fixed (with $|u| \leq \delta^{-1}$) and $|w| \geq K_2$, there is exactly one root of $D_w(z) - u(D_{w\alpha}(z) - (1-z))$ in the closed disk $\{z \mid |z| \leq \rho\}$.

When $u = 0$, this lemma implies (for $|w| \geq K_2$) that $D_w(z)$ has exactly one root in the disk $\{z \mid |z| \leq \rho\}$. Let A_w denote this root, and let $B_w = D'_w(A_w)$. Also, let $C_w(u)$ denote the root of $D_w(z) - u(D_{w\alpha}(z) - (1-z))$ in the closed disk $\{z \mid |z| \leq \rho\}$. Finally, we define

$$\begin{aligned} E_w(u) &:= (\partial_z(D_w(z) - u(D_{w\alpha}(z) - (1-z))))|_{z=C_w} \\ &= D'_w(C_w) - u(D'_{w\alpha}(C_w) + 1). \end{aligned}$$

We have precisely determined the singularities of $M(z,u)$. Next, we compare $M(z,u)$ to $M^I(z,u)$ to show that M_n and M_n^I have asymptotically similar behaviors.

E. Comparing Suffix Trees To Tries

We shall finally prove here Theorem 2 by comparing the generating functions $M(z,u)$ and $M^I(z,u)$. We define

$$Q(z,u) = M(z,u) - M^I(z,u).$$

Using the notation from (13) and (15), if we write

$$\begin{aligned} M^I_{w,\alpha}(z,u) &= \frac{u\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{1 - z(1 - \mathsf{Pr}(w))} \\ &\quad \times \frac{z\mathsf{Pr}(w)\mathsf{Pr}(\alpha)}{1 - z(1 + u\mathsf{Pr}(w)\mathsf{Pr}(\alpha) - \mathsf{Pr}(w))} \\ M_{w,\alpha}(z,u) &= \frac{u\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{D_w(z)} \\ &\quad \times \frac{D_{w\alpha}(z) - (1-z)}{D_w(z) - u(D_{w\alpha}(z) - (1-z))} \end{aligned}$$

then we have proved that

$$Q(z,u) = \sum_{\substack{w \in \mathcal{A}^* \\ \alpha \in \mathcal{A}}} (M_{w,\alpha}(z,u) - M^I_{w,\alpha}(z,u)).$$

We also define $Q_n(u) = [z^n]Q(z,u)$. We denote the contribution to $Q_n(u)$ from a specific w and α as $Q_n^{(w,\alpha)}(u) = [z^n](M_{w,\alpha}(z,u) - M^I_{w,\alpha}(z,u))$. Then we observe that

$$Q_n^{(w,\alpha)}(u) = \frac{1}{2\pi i} \oint (M_{w,\alpha}(z,u) - M^I_{w,\alpha}(z,u)) \frac{dz}{z^{n+1}}$$

where the path of integration is a circle about the origin with counterclockwise orientation.

We define

$$I_n^{(w,\alpha)}(\rho,u) = \frac{1}{2\pi i} \int_{|z|=\rho} (M_{w,\alpha}(z,u) - M^I_{w,\alpha}(z,u)) \frac{dz}{z^{n+1}}.$$

By Cauchy's theorem, we observe that the contribution to $Q_n(u)$ from a specific w and α is exactly

$$\begin{aligned} Q_n^{(w,\alpha)}(u) &= I_n^{(w,\alpha)}(\rho,u) - \mathrm{Res}_{z=A_w} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} \\ &\quad - \mathrm{Res}_{z=C_w(u)} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} \\ &\quad + \mathrm{Res}_{z=1/(1-\mathsf{Pr}(w))} \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}} \\ &\quad + \mathrm{Res}_{z=1/(1+u\mathsf{Pr}(w)\mathsf{Pr}(\alpha)} - \mathsf{Pr}(w)) \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}}. \end{aligned} \tag{19}$$

To simplify this expression, note that

$$\begin{aligned} \mathrm{Res}_{z=A_w} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} &= -\frac{\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{B_w} \frac{1}{A_w^{n+1}} \\ \mathrm{Res}_{z=C_w(u)} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} &= \frac{\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{E_w(u)} \frac{1}{C_w(u)^{n+1}} \\ \mathrm{Res}_{z=1/(1-\mathsf{Pr}(w))} \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}} &= \mathsf{Pr}(\beta)\mathsf{Pr}(w)(1 - \mathsf{Pr}(w))^n \\ \mathrm{Res}_{z=1/(1+u\mathsf{Pr}(w)\mathsf{Pr}(\alpha)-\mathsf{Pr}(w))} & \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}} \\ &= -\mathsf{Pr}(\beta)\mathsf{Pr}(w)(1 + u\mathsf{Pr}(w)\mathsf{Pr}(\alpha) - \mathsf{Pr}(w))^n. \end{aligned} \tag{20}$$

It follows from (19) that

$$\begin{aligned} Q_n^{(w,\alpha)}(u) &= I_n^{(w,\alpha)}(\rho,u) + \frac{\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{B_w} \frac{1}{A_w^{n+1}} \\ &\quad - \frac{\mathsf{Pr}(\beta)\mathsf{Pr}(w)}{E_w(u)} \frac{1}{C_w(u)^{n+1}} \\ &\quad + \mathsf{Pr}(\beta)\mathsf{Pr}(w)(1 - \mathsf{Pr}(w))^n \\ &\quad - \mathsf{Pr}(\beta)\mathsf{Pr}(w)(1 + u\mathsf{Pr}(w)\mathsf{Pr}(\alpha) - \mathsf{Pr}(w))^n. \end{aligned} \tag{21}$$

We next determine the contribution of the $z = A_w$ terms of $M(z,u)$ and the $z = 1/(1-\mathsf{Pr}(w))$ terms of $M^I(z,u)$ to the difference $Q_n(u) = [z^n](M(z,u) - M^I(z,u))$.

Lemma 4: The "A_w terms" and the "$1/(1-\mathsf{Pr}(w))$ terms" (for $|w| \geq K_2$) altogether have only $O(n^{-\epsilon})$ contribution to $Q_n(u)$, i.e.,

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \left(-\mathrm{Res}_{z=A_w} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} + \mathrm{Res}_{z=1/(1-\mathsf{Pr}(w))} \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}} \right) = O(n^{-\epsilon})$$

for some $\epsilon > 0$.

Proof: We define

$$f_w(x) = \frac{1}{A_w^{x+1}B_w} + (1-\mathsf{Pr}(w))^x$$

for x real. So by the set of equations in (20) it suffices to prove that

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \mathsf{Pr}(\beta)\mathsf{Pr}(w) f_w(x) = O(x^{-\epsilon}).$$

Note that

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \mathsf{Pr}(\beta)\mathsf{Pr}(w) f_w(x)$$

is absolutely convergent for all x. Also, $\bar{f}_w(x) = f_w(x) - f_w(0)e^{-x}$ is exponentially decreasing when $x \to +\infty$ and is $O(x)$ when $x \to 0$ (notice that we utilize the $f_w(0)e^{-x}$ term in order to make sure that $\bar{f}_w(x) = O(x)$ when $x \to 0$; this provides a fundamental strip for the Mellin transform in the next step). Therefore, its Mellin transform

$$\bar{f}^*_w(s) = \int_0^\infty \bar{f}_w(x) x^{s-1}\, dx$$

is wellde fined for $\Re(s) > -1$ (see [8] and [27]). We compute

$$\bar{f}^*_w(s) = \Gamma(s)\left(\frac{(\log A_w)^{-s} - 1}{A_w B_w} + (-\log(1-\mathsf{Pr}(w)))^{-s} - 1 \right)$$

where Γ denotes the Euler gamma function, and we note that

$$(\log A_w)^{-s} = \left(\frac{\mathsf{Pr}(w)}{S_w(1)}\right)^{-s} (1 + O(\mathsf{Pr}(w)))$$
$$(-\log(1-\mathsf{Pr}(w)))^{-s} = \mathsf{Pr}(w)^{-s}(1 + O(\mathsf{Pr}(w))).$$

Also

$$A_w = 1 + \frac{1}{S_w(1)}\mathsf{Pr}(w) + O(\mathsf{Pr}(w)^2),$$
$$B_w = -S_w(1) + \left(-\frac{2S'_w(1)}{S_w(1)} + m \right) \mathsf{Pr}(w) + O(\mathsf{Pr}(w)^2).$$

Therefore

$$\frac{1}{A_w B_w} = -\frac{1}{S_w(1)} + O(|w|\mathsf{Pr}(w)),$$

and

$$\bar{f}^*_w(s) = \Gamma(s)\left(\mathsf{Pr}(w)^{-s}\left(-S_w(1)^{s-1} + 1 + O(|w|\mathsf{Pr}(w))\right) + \frac{1}{S_w(1)} - 1 + O(|w|\mathsf{Pr}(w)) \right).$$

We define $g^*(s) = \sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \mathsf{Pr}(\beta)\mathsf{Pr}(w)\bar{f}^*_w(s)$. Then we compute

$$\begin{aligned} g^*(s) &= \sum_{\alpha\in\mathcal{A}} \mathsf{Pr}(\beta) \sum_{|w|\geq K_2} \mathsf{Pr}(w)\bar{f}^*_w(s) \\ &= \sum_{\alpha\in\mathcal{A}} \mathsf{Pr}(\beta)\Gamma(s) \sum_{m=K_2}^{\infty} \left(\sup\{q^{-\Re(s)}, 1\}\delta\right)^m \\ &= O(1) \end{aligned}$$

where the last equality is true because $1 \geq p^{-\Re(s)} \geq q^{-\Re(s)}$ when $\Re(s)$ is negative, and also because $q^{-\Re(s)} \geq p^{-\Re(s)} \geq 1$ when $\Re(s)$ is positive. We always have $\delta < 1$. Also, there exists $c > 0$ such that $q^{-c}\delta > 1$. Therefore, $g^*(s)$ is analytic in $\Re(s) \in (-1, c)$. Working in this strip, we choose ϵ with $0 < \epsilon < c$. Then we have

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \mathsf{Pr}(\beta)\mathsf{Pr}(w) f_w(x) = \frac{1}{2\pi i}\int_{\epsilon - i\infty}^{\epsilon + i\infty} g^*(s) x^{-s} + \sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \mathsf{Pr}(\beta)\mathsf{Pr}(w) f_w(0) e^{-x}.$$

Majorizing under the integral, we see that the first term is $O(x^{-\epsilon})$ since $g^*(s)$ is analytic in the strip $\Re(s) \in (-1, c)$ (and $-1 < \epsilon < c$). Also, the second term is $O(e^{-x})$. This completes the proof of the lemma. □

Now we bound the contribution to $Q_n(u)$ from the $C_w(u)$ terms of $M(z,u)$ and the $z = 1/(1 + u\mathsf{Pr}(w)\mathsf{Pr}(\alpha) - \mathsf{Pr}(w))$ terms of $M^I(z,u)$.

Lemma 5: The "$C_w(u)$ terms" and the "$1/(1 + u\mathsf{Pr}(w)\mathsf{Pr}(\alpha) - \mathsf{Pr}(w))$ terms" (for $|w| \geq K_2$) altogether have only $O(n^{-\epsilon})$ contribution to $Q_n(u)$, for some $\epsilon > 0$. More precisely

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} \left(-\mathrm{Res}_{z=C_w(u)} \frac{M_{w,\alpha}(z,u)}{z^{n+1}} + \mathrm{Res}_{z=1/(1+u\mathsf{Pr}(w)\mathsf{Pr}(\alpha)-\mathsf{Pr}(w))} \times \frac{M^I_{w,\alpha}(z,u)}{z^{n+1}} \right) = O(n^{-\epsilon}).$$

Proof: The proof technique is the same as the one for Lemma 4 above. □

Next we note that the $I_n^{(w,\alpha)}(\rho,u)$ terms in (21) have $O(n^{-\epsilon})$ contribution to $Q_n(u)$.

Lemma 6: The "$I_n^{(w,\alpha)}(\rho,u)$ terms" (for $|w| \geq K_2$) altogether have only $O(n^{-\epsilon})$ contribution to $Q_n(u)$, for some $\epsilon > 0$. More precisely

$$\sum_{\substack{|w|\geq K_2\\ \alpha\in\mathcal{A}}} I_n^{(w,\alpha)}(\rho,u) = O(n^{-\epsilon}).$$

Proof: We omit the proof here; see [28] for a proof. □

Finally, we consider the contribution to $Q_n(u)$ from small words $|w|$. Basically, we observe that $|w|$ has a normal distribution with mean $\frac{1}{h}\log n$ and variance $\theta \log n$, where $h = -p\log p - q\log q$ denotes the entropy of the source, and θ is a constant. Therefore, $|w| \leq K_2$ is extremely unlikely, and as a result, the contribution to $Q_n(u)$ from words w with $|w| \leq K_2$ is very small.

Lemma 7: The terms $\sum_{\substack{|w|<K_2 \\ \alpha\in\mathcal{A}}} (M_{w,\alpha}(z,u) - M^I_{w,\alpha}(z,u))$ altogether have only $O(n^{-\epsilon})$ contribution to $Q_n(u)$.

Proof: Again, we omit the proof due to space constraints. See [28]. □

All contributions to (21) have now been analyzed. We are finally prepared to summarize our results. Combining the last four lemmas, we see that $Q_n(u) = O(n^{-\epsilon})$ uniformly for $|u| \leq \delta^{-1}$, where $\delta^{-1} > 1$. For ease of notation, we define $b = \delta^{-1}$. Finally, we apply Cauchy's theorem again. We compute

$$\begin{aligned}\Pr(M_n = k) - \Pr(M_n^I = k) &= [u^k z^n]Q(z,u) \\ &= [u^k]Q_n(u) \\ &= \frac{1}{2\pi i}\int_{|u|=b}\frac{Q_n(u)}{u^{k+1}}\,du.\end{aligned}$$

Since $Q_n(u) = O(n^{-\epsilon})$, it follows that

$$\begin{aligned}|\Pr(M_n = k) - \Pr(M_n^I = k)| &\leq \frac{2\pi b}{|2\pi i|}\frac{O(n^{-\epsilon})}{b^{k+1}} \\ &= O(n^{-\epsilon}b^{-k}).\end{aligned}$$

Thus, Theorem 2 holds. It follows that M_n and M_n^I have asymptotically the same distribution, and therefore M_n and M_n^I asymptotically have the same factorial moments. The main result of [29] gives the asymptotic distribution and factorial moments of M_n^I. As a result, Theorem 2 follows immediately. Therefore, M_n follows the logarithmic series distribution, i.e., $\Pr(M_n = j) = \frac{p^j q + q^j p}{jh}$ (plus some small fluctuations if $\ln p/\ln q$ is rational). Theorem 1 is finally proved.

VI. Concluding Remarks

From the algorithmic perspective, two immediate challenges remain. First, we would like to make LZRS'77 on-line. The implementation of LZRS'77 described here is off-line because the blocks need to be processed backward, but it is not clear if this is absolutely necessary. Second, we would like to be able to protect the first block while maintaining backward compatibility. Note that we cannot embed the parity bits of the first block in the pointers of the last, because otherwise we would introduce a circular dependency in the process. From an analytic perspective, it would be interesting to extend Theorem 1 to Markov sources. While it is well-known [32] that the expectation for Markov sources is $\boldsymbol{E}[M_n] = O(1)$ (cf. [16]), not much is known about the distribution of M_n under that probabilistic model. The recent work of Fayolle and Ward [7], in which they extend the analysis of [11] to Markov sources, is a step in that direction.

Finally, we should point out that there is a way to extend our scheme to recover more than a constant number of redundant bits (and potentially to strongly mixing sources along the lines of [13]). One just has to give up the idea of always looking for the longest match and instead agree to use "long enough" matches. Such a scheme is still asymptotically optimal with the (compression) bit rate $1/h + O(\log\log n/\log n)$ and with M_n growing slowly with n. For example, instead of using the longest match we search for the rth longest match. We expect that if r grows with n in such a way that the rth longest match is of order $(\log n - \log\log n)/h$, then M_n grows with n (possibly $M_n = O(\log n)$?); in this case, only the constant of the asymptotic redundancy $O(\log\log n/\log n)$ is affected.

References

[1] L. V. Ahlfors, *Complex Analysis*. New York: McGraw-Hill, 1979.
[2] M. J. Atallah and S. Lonardi, "Augmenting LZ–77 with authentication and integrity assurance capabilities," *Concurrency and Computation: Practice and Experience*, vol. 6, pp. 1063–1076, 2004.
[3] T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*. Englewood Cliffs, NJ: Prentice-Hall, 1990.
[4] V. Castelli and L. Lastras-Montano, "Bounds on expansion in LZ'77-like coding," in *Proc. IEEE Int. Symp. Information Theory*, Chicago, IL, Jun./Jul. 2004, p. 58.
[5] P. Flajolet, "An average-case analysis of basic parameters of the suffix tree," in *Mathematics and Computer Science*, M. Drmota, D. Gardy, and B. Gittenberger, Eds. Vienna, Austria: Birkhäuser, 2004, pp. 217–227.
[6] E. R. Fiala and D. H. Greene, "Data compression with finite windows," *Commun. ACM*, vol. 32, no. 4, pp. 490–505, 1989.
[7] J. Fayolle and M. D. Ward, "Analysis of the average depth in a suffix tree under a Markov model," in *Proc. Int. Conf. Analysis of Algorithms*, Barcelona, Spain, 2005, pp. 95–104.
[8] P. Flajolet, X. Gourdon, and P. Dumas, "Mellin transforms and asymptotics: Harmonic sums," *Theor. Comp. Sci.*, vol. 144, pp. 3–58, 1995.
[9] R. N. Horspool, "The effect of nongreedy parsing in Ziv-Lempel compression methods," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, 1995, pp. 302–311.
[10] L. Guibas and A. M. Odlyzko, "Periods in strings," *J. Combin. Theory*, vol. 30, pp. 19–43, 1981.
[11] P. Jacquet and W. Szpankowski, "Autocorrelation on words and its applications: Analysis of suffix trees by string-ruler approach," *J. Comb. Theory*, vol. A66, pp. 237–269, 1994.
[12] P. Jacquet and W. Szpankowski, "Analytical depoissonization and its applications," *Theor. Comp. Sci.*, vol. 201, pp. 1–62, 1998.
[13] P. Jacquet, W. Szpankowski, and I. Apostol, "Universal predictor based on pattern matching," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1462–1472, Jun. 2002.
[14] P. Karn, General-purpose Reed-Solomon Encoder/Decoder, v4.0 2004 [Online]. Available: http://www.ka9q.net/code/fec
[15] D. E. Knuth, *Fundamental Algorithms*, 3rd ed. Reading, MA: Addison-Wesley, 1997.
[16] S. Lonardi and W. Szpankowski, "Joint source-channel LZ'77 coding," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, 2003, pp. 273–282.
[17] M. Lothaire, Ed., "Analytic Approach to Pattern Matching," in *Applied Combinatorics on Words*. Cambridge, U.K.: Cambridge Univ. Press, 2005, ch. 7.
[18] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: Elsevier, 1977.
[19] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. SIAM*, vol. 8, pp. 300–304, 1960.
[20] M. Régnier and W. Szpankowski, "On pattern frequency occurrences in a Markovian sequence," *Algorithmica*, vol. 22, pp. 631–649, 1998.
[21] Y. Reznik and W. Szpankowski, "On average redundancy rate of the Lempel-Ziv codes with k-error protocol," *Inf. Sci.*, vol. 135, pp. 57–70, 2001.
[22] M. Rodeh, V. R. Pratt, and S. Even, "Linear algorithm for data compression via string matching," *J. Assoc. Comput. Mach.*, vol. 28, no. 1, pp. 16–24, Jan. 1981.

[23] K. Sayood, H. Otu, and N. Demir, "Joint source/channel coding for variable length codes," *IEEE Trans. Commun.*, vol. 48, no. 5, pp. 787–794, May 2000.
[24] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*. Reading, MA: Addison-Wesley, 1996.
[25] J. Storer and J. Reif, "Error resilient optimal data compression," *SIAM J. Comput.*, vol. 26, pp. 934–949, 1997.
[26] W. Szpankowski, "A generalized suffix tree and its (un)expected asymptotic behaviors," *SIAM J. Comput.*, vol. 22, pp. 1176–1198, 1993.
[27] W. Szpankowski, *Average Case Analysis of Algorithms on Sequences*. New York: Wiley, 2001.
[28] M. D. Ward, "Analysis of the Multiplicity Matching Parameter in Suffix Trees," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, May 2005.
[29] M. D. Ward and W. Szpankowski, "Analysis of a randomized selection algorithm motivated by the LZ'77 scheme," in *Proc. 1st Workshop on Analytic Algorithmics and Combinatorics*, New Orleans, LA, 2004, pp. 153–160.
[30] M. D. Ward and W. Szpankowski, "Analysis of the multiplicity matching parameter in suffix trees," in *Proc. Int. Conf. Analysis of Algorithms*, Barcelona, Spain, 2005, pp. 307–322.
[31] Y. Wu, S. Lonardi, and W. Szpankowski, "Error-resilient LZW data compression," in *IEEE Data Compression Conf.*, Snowbird, UT, 2006, pp. 193–202.
[32] A. J. Wyner, "The redundancy and distribution of the phrase lengths of the fixed-database Lempel-Ziv algorithm," *IEEE Trans. Info. Theory*, vol. 43, no. 5, pp. 1439–1465, Sept. 1997.
[33] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 337–343, May 1977.