

Codelet Parsing: Quadratic-time, Sequential, Adaptive Algorithms for Lossy Compression

Dharmendra S. Modha
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120
email: dmodha@almaden.ibm.com

Abstract

We propose new algorithms, collectively termed, *codelet parsing*, for lossy compression. The algorithms *sequentially* parse a given source sequence into phrases, say, *sourcelets*, and map each sourcelet to a distorted phrase, say, a *codelet*, such that the per-letter distortion between the two phrases does not exceed the desired distortion. The algorithms *adaptively* maintain a codebook (a set of codewords), and do not require any *a priori* knowledge of the source statistics. The algorithms use approximate string matching and, as key new idea, at each epoch, carefully select one of the many approximately matching codewords to balance between the code rate in the current epoch versus the code rate from resulting codebooks in future epochs. The algorithms are *quadratic-time* in the length of the source sequence and output a distorted sequence that can be naturally losslessly compressed using the Lempel-Ziv (LZ78) algorithm.

1 Introduction

1.1 Motivation

We summarize the problem of lossy data compression at a fixed distortion level; for an extensive survey, see [1]. Suppose we are given a finite string $x_1^n \equiv x_1, x_2, \dots, x_n$ of length n drawn from a finite source alphabet B according to a source distribution P . We would like to find a distorted or lossy version of x_1^n , say, $y_1^n = y_1, y_2, \dots, y_n$, drawn from a finite reproduction alphabet \hat{B} such that the average single-letter distortion between the two strings is at most D (according to some bounded, non-negative distortion measure d) and that the lossy sequence y_1^n is highly compressible. We would like to minimize the *rate* at which y_1^n must be transmitted subject to the *distortion* constraint. The central result of rate-distortion theory is that for source sequences generated by a stationary, ergodic stochastic process, asymptotically (as $n \rightarrow \infty$), the *rate-distortion* function $R(D, P)$ is an achievable lower bound on the compression rate.

The central problem of lossy source coding is to find an universal (for stationary, ergodic sources), sequential, adaptive, and polynomial-time algorithm. The quest for such algorithms is important in theory as well in practice owing to broadband applications such as streaming multimedia, images, audio, cellular voice, and text. While such algorithms exist when $D = 0$, so far, no algorithm attaining all the desiderata is known when $D > 0$. Indeed, prominent researchers have attested to this fact, for example, [1, p. 2709] noted that “All universal lossy coding schemes found to date lack the relative simplicity that imbues Lempel-Ziv coders and arithmetic coders with economic viability. . . . This suggests it is unlikely that the “holy grail” of implementable universal lossy source coding will be discovered soon.”. Also, [2] noted that “...it is our belief that a universal lossy source coding scheme with attractive computational complexity aspects will never be found.” Finally, it was suggested that [3] “further investigations of suboptimal and practical heuristics for lossy compression are needed.”

1.2 Our Contribution

In this paper, we design quadratic-time, sequential, and adaptive Lempel-Ziv-type algorithms for lossy compression. At this stage of research, it is not known if our algorithms are universal for some classes of sources. All our algorithms share the same basic structure and differ only in one choice, and, hence, here, we refer to them as a single algorithm.

We now briefly describe the algorithm. Our algorithm is in the spirit of the incremental parsing Lempel-Ziv (LZ78) algorithm [4] for lossless coding, and, when no distortion is desired, the algorithm simplifies to LZ78. The algorithm *sequentially* parses the source sequence into phrases, say, *sourcelets*, and maps each sourcelet to a distorted phrase, say, a *codelet*, such that the per-letter distortion between the two phrases does not exceed the desired distortion. The algorithm *adaptively* maintains a codebook (a set of codewords¹) which contains all one-letter extensions of previously emitted codelets; hence, it generates and continually grows the codebook on-the-fly in response to the observed source sequence without any *a priori* knowledge of the source. Both the encoder and decoder maintain the same codebook. The algorithm carries out a sequential procedure by iterating the following steps: (i) given the current codebook find the set of all codewords that match (at a given distortion) the corresponding prefix of the unparsed source sequence; and (ii) from all these matching codewords carefully choose that which best balances between the per-letter code rate in the current epoch and the quality of the resulting codebook for future epochs. The key observation is that, typically, more than one approximately matching codewords will be found. Multiple matches are a sign of underlying redundancy, and, hence, are a symptom of the fact that we are not operating near the rate-distortion curve. We provide schemes for choosing between multiple matches. A key new idea is to examine the coding complexity of the source sequence (resp. the distorted sequence)

¹We use the term *codelet* to denote a previously emitted distorted phrase and *codeword* to denote a phrase that is a one-letter extension of a codelet and is itself not a codelet (yet).

given the distorted sequence (resp. the source sequence) as *side information*. To carry out this computation, we rely upon a version of the Lempel-Ziv algorithm for lossless source coding with side information [5]. Such judicious codeword selection is intended to iteratively improve the codebook quality. We think of such selection as a kind of “gold-washing” [6] or “natural-type selection” [7] to select good codewords, and refer to our selection mechanism as: *codelet parsing*. As an important side benefit, the algorithm outputs a distorted sequence that can be naturally losslessly compressed using the LZ78 algorithm.

1.3 Scope and Outline of the Paper

The scope of this paper is limited to theoretical formulation and presentation of the algorithms. Various empirical simulation results, refinements and extensions, and, finally, detailed discussion of data structure design will be presented in a forthcoming paper.

In Section 2, we describe various known algorithms and put our work in context. In Section 3, we present the necessary preliminaries. In Section 4, we recall LZ78 and a version of LZ78 with side information. In Section 5, we present the new algorithms.

2 Prior Work

We briefly (and non-exhaustively) review various known lossy coding schemes with a focus on algorithmic results. We will confine our discussion to finite (discrete) source and reproduction alphabets.

Cheung and Wei [8] extended the move-to-front algorithm to lossy source coding. The algorithm adaptively builds its codebook from source words. Since, generally, the optimal reproduction distribution differs from the source distribution, the algorithm is known to be sup-optimal [9]. Later, Zhang and Wei [6] proposed an on-line lossy coding algorithm for the fixed-rate case that uses a “gold-washing” mechanism for promoting frequently used codewords (during a time interval of specified length) to permanent status while randomly generating new candidate codewords. Their algorithm is universal for memoryless sources.

Another group of papers have focussed on lossy extensions of the Lempel-Ziv algorithm. The central idea is to use approximate string matching instead of exact string matching used in the Lempel-Ziv algorithms. Morita and Kobayashi [10] extended the LZW algorithm, say, ELZW. There are several key differences between codelet parsing and ELZW. At every step, ELZW finds the longest source phrase such that the source phrase and all its prefixes match at least one codeword and its corresponding prefixes within the desired distortion. We do not require such strict sequentiality, and, hence, can often find longer matches. Due to strong sequentiality, in certain cases, ELZW has the property that it transmits the first few letters of every phrase undistorted. Second, amongst multiple codeword matches, ELZW selects the one with the smallest distortion. Finally, like the LZW algorithm, after ELZW has found the longest matching source phrase and has selected a corresponding codeword,

it adds the newly selected codeword extended by the next source letter to the codebook. As a result, each codeword is composed by sampling different source letters and by concatenating them. At least in the memoryless case, the end effect is that each codeword is now drawn from the source distribution. As a result, the algorithm is known to be sub-optimal for memoryless sources [9]. In comparison, we add all one-letter extensions of the selected codeword to the codebook. From these multiple codewords, we adaptively pick the codewords that prove to be useful. Hence, the codewords in our algorithm are not all drawn from the source distribution.

Constantinescu and Storer [11, 12] combined ideas from lossless Lempel-Ziv algorithm and vector quantization to design first practical implementations of lossy image compression based on approximate pattern matching. The problem of “selecting amongst multiple matches” mentioned above was termed the “Match Heuristic” in their work; see, also, Storer [13, p. 111].

Later, Steinberg and Gutman [14] and Luczak and Szpankowski [3] considered the fixed-database version of the Lempel-Ziv algorithm, and provided sub-optimal performance guarantees. Finally, Yang and Kieffer [9] established that all previous fixed-database extensions of the Lempel-Ziv algorithm are suboptimal; intuitively, the mismatch between the distribution generating and fixed database (the so-called training sequence) and the optimal reproduction distribution causes these algorithms to be sub-optimal. Kontoyiannis [15] presented a scheme where multiple databases (each drawn according to a different reproduction distribution) are used at the encoder and must also be known to the decoder. When the reproduction alphabet is large, the number of training databases is unreasonably large. Atallah et al. [16] considered a cubic-time, adaptive algorithm (PMIC) in the spirit of LZ77. Their algorithm is not sequential since its encoding delay grows faster than $o(n)$. Alzina et al. [17] combined ideas from [16] and [11, 12] to propose a 2D-PMIC algorithm that is more suited for 2D images. We note that while we have developed codelet parsing in the framework of LZ78, it can be easily adapted to ELZW of [10] and LZ77-based PMIC of [16].

Continuing the quest for Lempel-Ziv-type lossy algorithms, Zamir and Rose [18] further studied the algorithm in [10]. From the multiple codewords that may match a source word, they suggest choosing one “at random”. From a theoretical perspective, Zamir and Rose [7] proposed a natural type selection scheme for finding the type of the optimal reproduction distribution. Their procedure can be thought of as a stochastic simulation of the Arimoto-Blahut algorithm for computing the rate-distortion function. In later work, Kochman and Zamir [19] pointed out that the theoretical procedure in [7] is in itself not practical and demonstrated an application of natural-type selection to on-line codebook selection from a parametric class.

Along a different line, Yang and Kieffer [2] have proposed exponential-time Lempel-Ziv-type block codes that are universal (for stationary, ergodic sources and for individual sequences). In a related work, Yang and Zhang [20] presented fixed-slope universal lossy coding schemes that search for the reproduction sequence through a trellis in a fashion reminiscent of the Viterbi algorithm. While finding the optimal reproduction sequence still takes exponential-time, the trellis structure allows computationally efficient heuristic codes that are also sequential in nature.

3 Preliminaries and Basic Definitions

All logarithms are base 2, that is, $\log \equiv \log_2$. The set of natural numbers is written as \mathbb{N} .

Let λ denote the empty string. Let A denote a finite alphabet. Let $|A|$ denote the number of symbols in A . We will refer to elements of A as *symbols* or *letters*. For $m \in \mathbb{N}$, let $A^m = \{(a_1, a_2, \dots, a_m) : a_i \in A, 1 \leq i \leq m\}$ denote the set of all sequences (strings, words, or phrases) of length m over A . By convention, $A^0 = \{\lambda\}$. Let $A^* = \cup_{m \geq 0} A^m$ denote the set of all sequences of finite length over A . For any string a of finite length, let $|a|$ denotes the length of a . For $i \leq j$, we let $a_i^j \equiv a_i, a_{i+1}, \dots, a_j$. If $j < i$, then a_i^j denotes the empty string λ . Let \circ denote the string concatenation operator.

We refer to a finite set of finite strings as a *dictionary*. A dictionary is termed *proper* if no string in the dictionary is a prefix of another. A dictionary is termed *complete* if every one-sided infinite sequence has a prefix in the dictionary.

Let B denote a *source* alphabet, and let \hat{B} denote a *reproduction* alphabet. We assume that both B and \hat{B} are finite. Let d denote a *distortion measure*, namely, a bounded, non-negative function on $B \times \hat{B}$. The distortion measure quantifies the loss of fidelity in representing a symbol in B using a symbol in \hat{B} . For example, for binary alphabets $B = \hat{B} = \{0, 1\}$, d may simply be the Hamming distance. We assume that $\max_{b \in B} \min_{\hat{b} \in \hat{B}} d(b, \hat{b}) = 0$. We now extend d to strings. For $m \in \mathbb{N}$, define the *single letter* distortion measure d_m on $B^m \times \hat{B}^m$ as $d_m(b_1^m, \hat{b}_1^m) = \frac{1}{m} \sum_{i=1}^m d(b_i, \hat{b}_i)$, where $b_1^m \in B^m$ and $\hat{b}_1^m \in \hat{B}^m$. For $m \in \mathbb{N}$, we say that a sequence $\hat{b}_1^m \in \hat{B}^m$ is a ϵ -*match*, $\epsilon \geq 0$, of a sequence $b_1^m \in B^m$ with respect to the single-letter distortion measure d_m , if $d_m(b_1^m, \hat{b}_1^m) \leq \epsilon$.

Practically, we are interested in the following problem. Given a source sequence $x_1^n \equiv x_1, x_2, \dots, x_n \in B^n$ and target distortion D , we would like to find a *reproduced sequence* or *distorted sequence* $y_1^n \equiv y_1, y_2, \dots, y_n \in \hat{B}^n$ such that $d_n(x_1^n, y_1^n) \leq D$ and that y_1^n is highly compressible.

4 Two Lossless Compression Algorithms

4.1 LZ78

We now describe LZ78 (also known as *incremental parsing*). Let U denote a finite alphabet. For $k \in \mathbb{N}$, let u_1^k denote a source sequence to be compressed. The idea is to append the sequence on the left side with the empty phrase, and, then, sequentially parse the resulting sequence into phrases as

$$u_1^k = \lambda \circ u_1^{\tau(1)} \circ u_{\tau(1)+1}^{\tau(2)} \circ \dots \circ u_{\tau(c_{LZ}(u_1^k)-1)+1}^{\tau(c_{LZ}(u_1^k))} \circ u_{\tau(c_{LZ}(u_1^k))+1}^k,$$

where every new phrase (except perhaps the last phrase) is the shortest prefix of the unparsed sequence that is distinct from all previously parsed phrases and $c_{LZ}(u_1^k)$ denotes the number of distinct phrases. Every phrase (including the last one) can

be encoded by pointing to its longest proper prefix amongst all previously parsed phrases, and by describing its last symbol. Hence, we can write the Lempel-Ziv codeword length function as:

$$L_{LZ}(u_1^k) = \sum_{i=1}^{c_{LZ}(u_1^k)+1} \log(i|U|).$$

For $u' \in U^*$, write $L_{LZ}(u'|u_1^k) = L_{LZ}(u_1^k \circ u') - L_{LZ}(u_1^k)$.

4.2 LZ78 with Side Information

We now describe a LZ78-type algorithm for data compression with side information [5]. Let U and V denote finite alphabets. For $k \in \mathbb{N}$, suppose we are given u_1^k and v_1^k , where u_1^k is the source sequence to be compressed without loss and v_1^k is the side information sequence that is also available to the decoder. Write $(uv)_i$ to mean the pair (u_i, v_i) , and $(uv)_i^j$ to mean (u_i^j, v_i^j) . Incrementally parse the joint sequence $(uv)_1^k$ using the LZ78 algorithm as

$$(uv)_1^k = \lambda \circ (uv)_{\tau(1)+1}^{\tau(1)} \circ (uv)_{\tau(1)+1}^{\tau(2)} \circ \dots \circ (uv)_{\tau(c_{LZ}((uv)_1^k)-1)+1}^{\tau(c_{LZ}((uv)_1^k))} \circ (uv)_{\tau(c_{LZ}((uv)_1^k)+1)}^k.$$

As a convention, we write $\tau(-1) = \tau(0) = 0$.

For $1 \leq i \leq c_{LZ}((uv)_1^k) + 1$, suppose that we have already transmitted $u_1^{\tau(i-1)}$ and $v_1^{\tau(i)}$. We show how to transmit $u_{\tau(i-1)+1}^{\tau(i)}$. Now, let $c((uv)_1^{\tau(i-1)}, v_{\tau(i-1)+1}^{\tau(i)-1})$ denote the number of previous uv phrases in $(uv)_1^{\tau(i-1)}$ whose v -part exactly equals $v_{\tau(i-1)+1}^{\tau(i)-1}$. Then, by definition of incremental parsing, one of these (uv) -phrases must be a proper prefix of $(uv)_{\tau(i-1)+1}^{\tau(i)}$. Hence, we can transmit roughly $\log c((uv)_1^{\tau(i-1)}, v_{\tau(i-1)+1}^{\tau(i)-1}) + \log |U|$ bits to completely specify $u_{\tau(i-1)+1}^{\tau(i)}$. Write the resulting code length as

$$L_{LZ}^s(u_1^k|v_1^k) = \sum_{i=1}^{c_{LZ}((uv)_1^k)+1} \left(\log c((uv)_1^{\tau(i-1)}, v_{\tau(i-1)+1}^{\tau(i)-1}) + \log |U| \right).$$

Similarly, define $L_{LZ}^s(v_1^k|u_1^k)$. For $u' \in U^*$ and $v' \in V^*$ such that $|u'| = |v'|$, write $L_{LZ}^s(u'|v', (uv)_1^k) = L_{LZ}^s(u_1^k \circ u'|v_1^k \circ v') - L_{LZ}^s(u_1^k|v_1^k)$.

Now, consider the sequence v_1^k that is parsed according to the joint sequence $(uv)_1^k$,

$$v_1^k = \lambda \circ v_1^{\tau(1)} \circ v_{\tau(1)+1}^{\tau(2)} \circ \dots \circ v_{\tau(c_{LZ}(v_1^k)-1)+1}^{\tau(c_{LZ}((uv)_1^k))} \circ v_{\tau(c_{LZ}((uv)_1^k)+1)}^k.$$

This is an interesting parsing of v_1^k . It has the property that a v -phrase can occur only after each of its proper prefix has appeared at least once. Also, a v -phrase (that is not the last phrase) of length ℓ can appear no more than $|B|^\ell$ -times (where B is the alphabet for the u -sequence). We now describe a scheme for compressing v_1^k given the incremental parsing of $(uv)_1^k$ imposed on v_1^k . At any time, we assign a normalized

weight to all previous v -phrases that corresponds to the number of times a particular phrase has appeared. Let w_i denote the weight function at time i .

$$L_w(v_1^k; (uv)_1^k) = \sum_{i=1}^{c_{LZ}((uv)_1^k)+1} \left(\log w_i(v_{\tau(i-1)+1}^{\tau(i)-1}) + \log |V| \right).$$

For $u' \in U^*$ and $v' \in V^*$ such that $|u'| = |v'|$, write $L_w(v'|v_1^k; (uv)_1^k \circ (u'v')) = L_w(v_1^k \circ v'; (uv)_1^k \circ (u'v')) - L_w(v_1^k; (uv)_1^k)$.

5 Codelet Parsing

5.1 A Generic Structure

Intuitively, the generic structure of our lossy sequential code is as follows. Given a source sequence $x_1^n \in B^n$, the code reads a variable-sized prefix of x_1^n , say, a source phrase, and outputs an equi-length distorted phrase such that the average per-symbol distortion between the two phrases is no more than D . The code then transitions to a new state. In this state, once again it repeats the whole sequence of events starting from the (as yet) unprocessed source sequence, and so on until the source sequence is exhausted. We can write a generic such parsing as follows. Append the source sequence on the left with the empty string, and, then sequentially parse the source sequence x_1^n into phrases, namely, *sourcelets*, as

$$x_1^n = \lambda \circ x_1^{\hat{\tau}(1)} \circ x_{\hat{\tau}(1)+1}^{\hat{\tau}(2)} \circ \dots \circ x_{\hat{\tau}(\hat{c}(x_1^n))-1+1}^{\hat{\tau}(\hat{c}(x_1^n))} \circ x_{\hat{\tau}(\hat{c}(x_1^n))+1}^n,$$

where quantities $\hat{c}(x_1^n)$ and $\hat{\tau}(1), \hat{\tau}(2), \dots, \hat{\tau}(\hat{c}(x_1^n))$ will be appropriately defined below. We will then construct the distorted sequence by mapping each sourcelet to a distorted phrase, namely, a *codelet*, as

$$y_1^n = \lambda \circ y_1^{\hat{\tau}(1)} \circ y_{\hat{\tau}(1)+1}^{\hat{\tau}(2)} \circ \dots \circ y_{\hat{\tau}(\hat{c}(x_1^n))-1+1}^{\hat{\tau}(\hat{c}(x_1^n))} \circ y_{\hat{\tau}(\hat{c}(x_1^n))+1}^n, \quad (1)$$

where we require that

C.1 Each codelet is a D -match of the corresponding sourcelet.

C.2 Each codelet (except perhaps the last one) is the shortest phrase that is not one of the previously parsed codelets, and, hence, (1) is incremental parsing of y_1^n .

Observe that, trivially, from C.1 and C.2, y_1^n is a D -match of x_1^n . Also, it follows from C.2 that y_1^n can be naturally losslessly transmitted using LZ78. In fact, as soon as a new y -phrase is produced, it can be immediately sent.

We think of the code as going through epochs. At epoch 0, the code reads sourcelet λ and emits codelet λ , at epoch 1, the code reads $x_1^{\hat{\tau}(1)}$ and emits $y_1^{\hat{\tau}(1)}$, and so forth and so on. For convenience, we write $\hat{\tau}(-1) = \hat{\tau}(0) = 0$. We introduce the notion of a *codebook* or a *dictionary*, namely, a set of *codewords*. Intuitively, a codeword is a one-letter extension of a previously emitted codelet and is itself not yet a codelet. At

epoch i , $i \geq 1$, we will use a codebook, S_i , that is obtained by taking all one-letter extensions of all previous codelets, namely, all phrases of the form $u \circ \hat{b}$ where u is one of the previously emitted codelets and $\hat{b} \in \hat{B}$. Also, all previous codelets are deleted from S_i . Codewords in the codebook S_i are not all drawn from the source distribution. The codebook S_i can be organized in a tree whose internal nodes constitute all previously emitted codelets and whose leaf nodes constitute all codewords that are available for future use. The set S_i is a complete and proper dictionary. We say that the codebook captures the *state* of the lossy sequential code. The set of all states is simply the set of all complete and proper dictionaries over \hat{B} .

Algorithm

Initially, set $i = 1$ and set the codebook $S_1 = \hat{B}$. Also, set $\hat{\tau}(0) = \hat{\tau}(-1) = 0$.
 while (the source sequence is not exhausted, that is, $(\hat{\tau}(i-1) < n)$)

1. Let C_i denote the set of codewords in S_i that D -match the corresponding prefix of $x_{\hat{\tau}(i-1)+1}^n$. From this set, choose one codeword as the codelet.
2. Compute $\hat{\tau}(i)$ by adding the length of the chosen codeword to $\hat{\tau}(i-1)$. Define the next sourcelet as $x_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i)}$. Define the next codelet $y_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i)}$ as the chosen codeword. Construct the new codebook by deleting the chosen codeword from S_i and by adding all one-letter extensions of the chosen codeword. Set $i = i + 1$.

endwhile

From an implementation perspective, finding the set C_i is reasonably easy. Organize the codebook S_i as a tree. Traverse the tree in a depth-first fashion and at every leaf compare the codeword corresponding to the leaf to the source phrase of the same length. Add any D -matching codeword to the set C_i . Total amount of computation required is proportional to the length of the string processed so far, and, hence, the total overall computation is at most quadratic in the worst case. A finer argument which takes into account the average phrase lengths can show that the total overall computation is actually $O(n^2/\log n)$. This computation bounds do not take into account the computation that may be required for choosing the codeword in Step 1.

5.2 Practical Algorithms

The above algorithm is missing one critical detail, namely, which codeword is selected in Step 1 from amongst multiple competing codewords. The very essence of the code lies in this choice. We now consider various practical algorithms for this choice.

The first idea [13, p. 112] is to choose the longest codeword from the set C_i in Step 1 of the Algorithm (ties are broken by selecting the most recent match and further ties are broken by selecting the lowest distortion match).

As a second idea, let C_i^l denote the set of longest codewords in C_i . We can select the next codelet as:

$$y_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i)} = \arg \min_{v \in C_i^l} \left[\underbrace{L_w(v|y_1^{\hat{\tau}(i-1)}; (xy)_1^{\hat{\tau}(i-1)} \circ (uv))}_{F_1} - \underbrace{L_{LZ}^s(v|u, (xy)_1^{\hat{\tau}(i-1)})}_{F_2} \right], \quad (2)$$

where $u = x_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i-1)+|v|}$. The intuition is that first by restricting attention to C_i^L we minimize the per-symbol code rate in the current epoch. The choice in (2) is intended to improve the quality of the codebook. We think of $F_1 - F_2$ as an *empirical estimate of mutual information*.

Finally, we can consider the following choice of the next codelet where we do not restrict attention to the longest codewords. Select $y_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i)} =$

$$\arg \min_{v \in C_i} \left[\underbrace{\frac{L_{LZ}(v|y_1^{\hat{\tau}(i-1)})}{|v|}}_{I_1} + \underbrace{\frac{L_w(v|y_1^{\hat{\tau}(i-1)}; (xy)_1^{\hat{\tau}(i-1)} \circ (uv))}{|v|}}_{I_2} - \underbrace{\frac{L_{LZ}^s(v|u, (xy)_1^{\hat{\tau}(i-1)})}{|v|}}_{I_3} \right], \quad (3)$$

where $u = x_{\hat{\tau}(i-1)+1}^{\hat{\tau}(i-1)+|v|}$. We think of the term I_1 as minimizing the per-letter code rate in the current epoch while the terms I_2 and I_3 as improving the quality of the codebook for future use.

Using careful data structure design, all of the above three choices can be implemented in quadratic-time. In all three cases, the amount of look-ahead from $x_{\hat{\tau}(i-1)}$ is exactly equal to the length of the longest codeword in S_i , and, hence, cannot be larger than $O(\sqrt{\hat{\tau}(i-1)})$. Hence, the algorithms are sequential with vanishing look-ahead. We will present empirical simulation studies comparing the three algorithms in a forthcoming paper, where we will also present several refinements and extensions. The most interesting open question is whether the last two algorithms can be theoretically shown to be universal for some classes of sources.

Acknowledgements. I am grateful to Corneliu Constantinescu, Robert Hecht-Nielsen, and Brian Marcus for valuable discussions.

References

- [1] T. Berger and J. D. Gibson, "Lossy source coding," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2693–2723, 1998.
- [2] E.-H. Yang and J. C. Kieffer, "Simple universal lossy data compression schemes derived from the Lempel-Ziv algorithm," *IEEE Trans. Inform. Theory*, vol. 42, no. 1, pp. 239–245, 1996.
- [3] T. Luczak and W. Szpankowski, "A suboptimal lossy data compression based on approximate pattern matching," *IEEE Trans. Inform. Theory*, vol. 43, pp. 1439–1451, 1997.
- [4] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inform. Theory*, vol. 24, no. 5, pp. 530–536, 1978.
- [5] J. Ziv, "Universal decoding for finite-state channels," *IEEE Trans. Inform. Theory*, vol. 31, no. 4, pp. 453–460, 1985.

- [6] Z. Zhang and V. K. Wei, "An on-line universal lossy data compression algorithm via continuous codebook refinement—part i: Basic results," *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 803–821, 1996.
- [7] R. Zamir and K. Rose, "Natural type selection in adaptive lossy compression," *IEEE Trans. Inform. Theory*, vol. 47, no. 1, pp. 99–111, 2001.
- [8] K. Cheung and V. K. Wei, "A locally adaptive source coding scheme," in *Bilkent Conf. on New Trends in Comm., Cont., and Signal Proc.*, pp. 1473–1482, 1990.
- [9] E.-H. Yang and J. C. Kieffer, "On the performance of data compression algorithms based upon string matching," *IEEE Trans. Inform. Theory*, vol. 44, pp. 47–65, 1998.
- [10] H. Morita and K. Kobayashi, "An extension of LZW coding algorithm to source coding subject to a fidelity criterion," in *Proc. 4th Joint Swedish-Soviet Int. Workshop on Information Theory, Gotland, Sweden*, pp. 105–109, 1989.
- [11] C. Constantinescu and J. A. Storer, "On-line adaptive vector quantization with variable size codebook entries," in *Data Compression Conf.*, pp. 32–41, 1993.
- [12] C. Constantinescu and J. A. Storer, "Improved techniques for single-pass vector quantization," *Proceedings of the IEEE*, vol. 82, no. 6, pp. 933–939, 1994.
- [13] J. A. Storer, *Data Compression: Methods and Theory*. Rockville, Maryland: Computer Science Press, 1988.
- [14] Y. Steinberg and M. Gutman, "An algorithm for source coding subject to a fidelity criterion, based on string matching," *IEEE Trans. Inform. Theory*, vol. 39, no. 3, pp. 877–886, 1993.
- [15] I. Kontoyiannis, "An implementable lossy version of the Lempel-Ziv algorithm—part i: Optimality for memoeyless sources," *IEEE Trans. Inform. Theory*, vol. 45, no. 7, pp. 2293–2305, 1999.
- [16] M. Atallah, Y. Genin, and W. Szpankowski, "Pattern matching image compression: Algorithmic and empirical results," in *Proc. Int. Conf. Image Processing, Lausanne, Switzerland*, vol. II, pp. 349–352, 1996.
- [17] M. Alzina, W. Szpankowski, , and A. Grama, "2D-pattern matching image and video compression," in *Data Compression Conf.*, pp. 424–433, 1999.
- [18] R. Zamir and K. Rose, "Towards lossy Lempel-Ziv: Natural type selection," in *Proc. Inform. Theory Workshop, Haifa, Israel*, p. 58, 1996.
- [19] Y. Kochman and R. Zamir, "Adaptive parametric vector quantization by natural type selection," in *Data Compression Conference*, pp. 392–401, 2002.
- [20] E.-H. Yang and Z. Zhang, "Variable-rate trellis source encoding," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 586–608, 1999.