Universal variable-length data compression of binary sources using fountain codes

Giuseppe Caire, Institut Eurecom
Shlomo Shamai, Technion
Amin Shokrollahi, EPFL
Sergio Verdú, Princeton University
giuseppe.caire@eurecom.fr, sshlomo@ee.technion.ac.il,amin.shokrollahi@epfl.ch,verdu@princeton.edu

Abstract — **This paper proposes a universal variable-length lossless compression algorithm based on fountain codes. The compressor concatenates the Burrows-Wheeler block sorting transform (BWT) with a fountain encoder, together with the closed-loop iterative doping algorithm. The decompressor uses a Belief Propagation algorithm in conjunction with the iterative doping algorithm and the inverse BWT. Linear-time compression/decompression complexity and competitive performance with respect to state-of-the-art compression algorithms are achieved.**

I. Introduction

It is known that *linear* fixed-length encoding can achieve for asymptotically large blocklength the minimum compression rate for memoryless sources [1] and for arbitrary (not necessarily stationary/ergodic) sources [2].

After initial attempts [3, 4, 5] to construct linear lossless codes were nonuniversal, limited to memoryless sources and failed to reach competitive performance with standard data compression algorithms, the interest in linear data compression waned. Recently [6, 7, 8] came up with a universal lossless data compression algorithm based on irregular low-density parity-check codes which has linear encoding and decoding complexity, can exploit source memory and in the experiments for binary sources presented in [6, 7, 8] showed competitive performance with respect to standard compressors such as `gzip`, PPM and `bzip`.

The scheme of [6, 7, 8] was based on the important class of sparse-graph error correcting codes called low-density parity check (LDPC) codes. The block-sorting transform (or Burrows-Wheeler transform (BWT)) [9] is a one-to-one transformation, which performs the following operation: it generates all cyclic shifts of the given data string and sorts them lexicographically. The last column of the resulting matrix is the BWT output from which the original data string can be recovered, knowing the *BWT index* which is the location of the original sequence in the matrix. The BWT shifts redundancy in the memory to redundancy in the marginal distributions. The redundancy in the marginal distributions is then much easier to exploit at the decoder as the decoding complexity is independent of the complexity of the source model (in particular, the number of states for Markov sources). The output of the BWT (as the blocklength grows) is asymptotically piecewise i.i.d. for stationary ergodic tree sources. The length, location, and distribution of the i.i.d. segments depend on the statistics of the source. The existing universal BWT-based methods for data compression generally hinge on the idea of compression for a memoryless source with an adaptive procedure that learns implicitly the local distribution of the piecewise i.i.d. segments, while forgetting the effect of distant symbols.

In the data compression algorithm of [6, 7], the compression is carried out by multiplication of the Burrows-Wheeler Transform of the source string with the parity-check matrix of an error correcting code. Of particular interest are LDPC codes since the belief propagation (BP) decoder is able to incorporate the time-varying marginals at the output of the BWT in a very natural way. The nonidentical marginals produced at the output of the BWT have a synergistic effect with the BP algorithm which is able to iteratively exploit imbalances in the reliability of variable nodes. The universal implementation of the algorithm where the encoder identifies the source segmentation and describes it to the decompressor is discussed in [8].

An important ingredient in the compression scheme of [6, 7, 8] is the ability to do decompression at the compressor. This enables to tune the choice of the codebook to the source realization and more importantly it enables the use of the Closed-Loop Iterative Doping (CLID) algorithm of [6, 2]. This is an efficient algorithm which enables zero-error variable-length data compression with performance which is quite competitive with that of standard data compression algorithms.

In this paper, instead of adopting irregular low-density parity check codes of a given rate approximately matched to the source we adopt a different approach based on rateless *fountain codes*. This class of codes turns out to be more natural for variable-length data compression applications than standard block codes and achieves in general comparable performance to the LDPC-based scheme of [6, 7, 8].

The rest of the paper is organized as follows. Section II reviews the main features of fountain codes for channel coding. Section III gives a brief summary of the principle of belief propagation decoding which is common to both channel and source decoding. Our scheme for data compression with fountain codes is explained in detail in Section IV in the setting of nonuniversal compression of binary sources. For further background on linear codes for data compression and the closed-loop iterative doping algorithm the reader is referred to [2]. The modelling module necessary for universal application is discussed in Section V. Section VI shows several experiments and comparisons of redundancy with off-the-shelf data compression algorithms run with synthetic sources. Throughout the discussion we limit ourselves to binary sources. The generalization to nonbinary alphabets is treated in [10].

II. Fountain Codes for Channel Coding

Fountain codes [11] form a new class of sparse-graph codes designed for protection of data against noise in environments where the noise level is not known a-priori. To achieve this, a fountain code produces a potentially limitless stream of output symbols for a given vector of input symbols. In practical applications, each output symbol is a linear function of the input symbols, and the output symbols are generated independently and randomly, according to a distribution which is

0-7803-8720-1/04/$20.00 ©2004 IEEE

chosen by the designer. The sequence of linear combinations of input symbols is known at the decoder. For example, they can be generated by pseudorandom generators with identical seeds.

Unlike traditional codes for which the code performance is measured in terms of the error rate as a function of the signal to noise ratio, the performance of fountain codes with respect to a given decoding algorithm is measured in terms of the error rate as a function of the reception *overhead*. The overhead of the decoding algorithm for a fountain code over a given communication channel is measured as the fraction of additional output symbols necessary to achieve the desired residual error rate. Here, the phrase "additional" is meant to be with respect to the Shannon limit of the underlying channel.

The decoding process for fountain codes is as follows: the receiver collects output symbols and estimates for each received output symbol the amount of information in that symbol. For example, when output symbols are bits, this measure of information can be obtained from the log-likelihood ratio (LLR) of the received bit. If this ratio is λ, then the empirical mutual information between the information symbol and its LLR equals $1 - h(p)$, where $p = 1/(1 + \exp(\lambda))$. The decoder stops collecting output bits as soon as the accumulated information carried by the observed channel outputs exceeds $(1+\omega)k$, where ω is the overhead associated with the fountain code, and k is the number of input symbols. Then the decoder uses its BP decoding algorithm to recover the input symbols from the information contained in the output symbols.

If the amount of information in the received output symbols is unknown but the channel is known to be memoryless stationary with capacity $C(\lambda)$ parameterized in the single output LLR λ, then decoding of fountain codes can be accomplished as follows: the decoder starts with an optimistic guess λ_1 of the channel parameter, collects an appropriate number of output symbols n_1 such that $k/n_1 = C(\lambda_1) - \delta$, where $\delta > 0$ is some positive rate margin that enforces successful decoding with high probability, and applies BP decoding based on the guess λ_1. In case the BP decoder is unsuccessful, a predetermined number of additional output symbols is collected such that the total number of output symbols is n_2, and the BP decoding is applied to the new graph using LLR $\lambda_2 = C^{-1}(k/n_2 + \delta)$. In case the BP decoder is unsuccessful, the same process is repeated using $n_3 > n_2$ output symbols and so on, until decoding is successful. In practice, instead of resetting the BP decoder at each new decoding attempt and especially if the initial guess λ_1 is likely to be not far from the true channel parameter, it might be more convenient to keep the same BP decoder running while incorporating the additional collected output symbols.

Discovered by Michael Luby [11], LT-codes are one of the first classes of efficient fountain codes for the erasure channel. An extension of this class of codes is the class of Raptor codes [13]. These classes of codes are very well suited for solving the compression problem, because the compression algorithm translates to a channel coding problem for a discrete memoryless channel with time-varying transition probability matrix which depends on the source. In applications, it is undesirable to tune the choice of the code to the sequence of transition probabilities, as in universal applications they are not known beforehand. In this case a fountain code is more amenable for universal implementation than other classes of efficient channel codes, such as irregular LDPC codes [22], mainly because a single code can be used regardless of the source rate.

For ease of exposition we concentrate on binary fountain codes. Let k be a positive integer, and let $\mathcal{D}$ be a distribution on $\mathbf{F}_2^k$. A Fountain Code ensemble with parameters $(k, \mathcal{D})$ has as its domain the space $\mathbf{F}_2^k$ of binary strings of length k, and as its target space the set of all sequences over $\mathbf{F}_2$, denoted by $\mathbf{F}_2^{\mathbf{N}}$. Formally, a Fountain Code ensemble with parameters $(k, \mathcal{D})$ is a linear map $\mathbf{F}_2^k \to \mathbf{F}_2^{\mathbf{N}}$ represented by an $\infty \times k$ matrix where rows are chosen independently with identical distribution $\mathcal{D}$ over $\mathbf{F}_2^k$. The blocklength of a Fountain Code is potentially infinite, but in our data compression applications we will solely consider truncated Fountain Codes with a number of coordinates which is tailored to the source realization.

The symbols produced by a Fountain Code are called *output symbols*, and the k symbols from which these output symbols are calculated are called *input symbols*. The input and output symbols could be elements of $\mathbf{F}_2$, or more generally the elements of any finite dimensional vector space over $\mathbf{F}_2$. In this paper, we are primarily concerned with Fountain Codes over the field $\mathbf{F}_2$, in which symbols are bits.

A special class of Fountain Codes is furnished by LT-Codes [11]. In this class, the distribution $\mathcal{D}$ has a special form described in the following. Let $(\Omega_1, \ldots, \Omega_k)$ be a distribution on $\{1, \ldots, k\}$ so that Ω_i denotes the probability that the value i is chosen. Often we will denote this distribution by its generator polynomial $\Omega(x) = \sum_{i=1}^{k} \Omega_i x^i$. The distribution $\Omega(x)$ induces a distribution on $\mathbf{F}_2^k$ in the following way: For any vector $v \in \mathbf{F}_2^k$ the probability of v is $\Omega_w / \binom{k}{w}$, where w is the Hamming weight of v. Abusing notation, we will denote this distribution in the following by $\Omega(x)$ again. An LT-code is a Fountain code with parameters $(k, \Omega(x))$.

The decoding graph of length n of a fountain code with parameters $(k, \Omega(x))$ is a bipartite graph with k nodes on one side (called *input nodes*, corresponding to the input symbols) and n nodes on the other side (called *output nodes*). The output nodes correspond to n output symbols collected at the output of the channel. The decoding graph is the *Tanner graph* of the linear encoder $\mathbb{F}_2^k \to \mathbb{F}_2^n$ obtained by restricting the fountain encoder mapping to those n components actually observed at the output.

A raptor code [13] performs a pre-coding operation with a linear code (e.g., an LDPC code) prior to using an LT-code. Without a pre-coder the average degree of an LT-coder needs to grow at least logarithmically in the length of the input to guarantee a small error probability, since otherwise there would exist input symbols that are not present in any of the linear equations generating the output symbols. The pre-code lowers the error floor present in an LT-code with small average degree.

III. Belief Propagation Decoding

In this section we give a description of the BP algorithm that is used in the decoding process of LT codes. The algorithm proceeds in rounds. In every round messages are passed from input nodes to output nodes, and then from output nodes back to input nodes. The message sent from the input node ι to the output node ω in the ℓth round of the algorithm is denoted by $\mathtt{m}_{\iota,\omega}^{(\ell)}$, and similarly the message sent from an output node ω to an input node ι is denoted by $\mathtt{m}_{\omega,\iota}^{(\ell)}$. These messages

are scalars in $\overline{\mathbf{R}} := \mathbf{R} \cup \{\pm\infty\}$. We will perform additions in this set according to the following rules: $a + \infty = \infty$ for all $a \neq -\infty$, and $a - \infty = -\infty$ for all $a \neq \infty$. The values of $\infty - \infty$ and $-\infty + \infty$ are undefined. Moreover, $\tanh(\infty/2) := 1$, and $\tanh(-\infty/2) := -1$.

Every value Y received from the channel has a log-likelihood ratio defined as $\ln(\Pr[Y = 0]/\Pr[Y = 1])$.

For every output node ω, we denote by Z_ω the corresponding log-likelihood ratio.

In round 0 of the BP algorithm the output nodes send to all their adjacent input nodes the value 0. Thereafter, the following update rules are used to obtain the messages passed at later rounds:

$$\tanh\left(\frac{\mathbf{m}_{\omega,\iota}^{(\ell)}}{2}\right) := \tanh\left(\frac{Z_\omega}{2}\right) \cdot \prod_{\iota' \neq \iota} \tanh\left(\frac{\mathbf{m}_{\iota',\omega}^{(\ell)}}{2}\right), \quad (1)$$

$$\mathbf{m}_{\iota,\omega}^{(\ell+1)} = \sum_{\omega' \neq \omega} \mathbf{m}_{\omega',\iota}^{(\ell)}, \quad (2)$$

where the product is over all input nodes adjacent to ω other than ι, and the sum is over all output nodes adjacent to ι other than ω, and $\ell \geq 0$.

After running the BP-algorithm for m rounds, the log-likelihood of each input symbol associated to node ι can be calculated as the sum $\sum_\omega \mathbf{m}_{\omega,\iota}^{(m)}$, where the sum is over all the output nodes ω adjacent to ι. In cases where the pre-code of the Raptor code is decoded separately from its LT-code, one may gather the log-likelihood of the input symbols, and run a decoding algorithm for the pre-code (which may itself be the BP algorithm). In that case, the prior log-likelihoods of the inputs are set to be equal to the calculated log-likelihoods according to (1).

In data compression applications, the CLID algorithm introduced in [6] runs BP at the encoder and supplies to the decoder the value of the lowest reliability symbol at certain iterations until successful decoding is achieved. Since the decoder runs an identical copy of the BP iterations, it knows the identity of the lowest reliability symbol, without the need to communicate this information explicitly. The symbols supplied by the CLID algorithm are referred to as doped symbols.

IV. Data Compression with Fountain Codes

We calculate from the input symbols $(x_1, \ldots, x_k)$ a vector of *intermediate symbols* $(y_1, \ldots, y_k)$ through a linear invertible $k \times k$ matrix $\mathbf{G}$:

$$(y_1, \ldots, y_k)^T = \mathbf{G}^{-1}(x_1, \ldots, x_k)^T \quad (3)$$

Next, using the intermediate symbols we calculate m output symbols $(x_{k+1}, \ldots, x_{k+m})$ according to a distribution $\Omega(x)$. [1] These m output symbols, together with the doped symbols obtained from the closed-loop iterative doping algorithm constitute the output of the compressor; hence, their total number should be as close as possible to $kh(p)$.

The Tanner graph on which the BP is run is a bipartite graph with k *intermediate nodes* (corresponding to the intermediate symbols) on one side and k input and m output nodes, corresponding to the k input and m output symbols, respectively) on the other side. The invertible matrix $\mathbf{G}$ is chosen as a random realization so that the degree distribution of the input nodes is $\Omega(x)$. [2] In this way the degree distribution of both the input and output nodes is equal to $\Omega(x)$. Notice that the resulting graph can be *interpreted* as the decoding graph of a $(k, \Omega(x))$ LT code with input symbols $(y_1, \ldots, y_k)$ and output symbols $(x_1, \ldots, x_{k+m})$, observed through a non-stationary BSC with transition probability p over the first k components and 0 over the second m components. The initial reliabilities (absolute value of the initial LLRs) of $(y_1, \ldots, y_k)$, $(x_1, \ldots, x_k)$ and $(x_{k+1}, \ldots, x_{k+m})$ are 0, $|\log((1-p)/p)|$ and $+\infty$, respectively.

Even though the matrix $\mathbf{G}$ is sparse, its inverse is generally dense. This has two consequences: the intermediate symbols behave like random coin flips, and the computation of (3) is, in principle, quadratic in k. In order to compute (3) in linear time, $\mathbf{G}$ should be such that the linear system $\mathbf{G}(y_1, \ldots, y_k)^T = (x_1, \ldots, x_k)^T$ can be solved by direct elimination of one unknown at a time. Equivalently, the BEC message-passing decoding algorithm applied to the Tanner graph of the parity equation $\mathbf{G}(y_1, \ldots, y_k)^T + (x_1, \ldots, x_k)^T = 0$ must terminate (i.e., it recovers all intermediate symbols $(y_1, \ldots, y_k)$).

We summarize the compression and decompression steps as follows:

1. After block sorting the original k-data vector, we obtain a block of symbols $(x_1, \ldots, x_k)$.
2. An intermediate block $(y_1, y_2, \ldots, y_k)$ of symbols is calculated by (3) (in practice, this is accomplished via message-passing).
3. A vector of m symbols $(x_{k+1}, \ldots, x_{k+m})$ is generated from $(y_1, \ldots, y_k)$ through encoding with an LT-code with parameters $(k, \Omega(x))$. Together with the doped bits generated as indicated below, $(x_{k+1}, \ldots, x_{k+m})$ forms the payload of the compressed data.
4. A bipartite graph is set up between the nodes corresponding to $(y_1, \ldots, y_k)$, and the nodes corresponding to $(x_1, \ldots, x_{k+m})$. The edges in this graph are defined as follows: for all i, there is an edge from x_i to all the bits among $(y_1, \ldots, y_k)$ of which x_i is the addition.
5. The BP algorithm is applied to the graph created in the previous step. The objective of this BP algorithm is to decode the symbols $(y_1, \ldots, y_k)$ using the full knowledge of the symbols $(x_{k+1}, \ldots, x_{k+m})$ and the a priori marginal source probabilities $\{P(x_i = 1) = p_i : i = 1, \ldots, k\}$. For $i = 1, \ldots k$, the LLR of the bit x_i (unavailable to the decoder) is set to $\log((1-p_i)/p_i)$. The marginal probabilities p_i are either known (in a non-universal setting) or estimated from the source sequence itself by the source modeler illustrated in Section V. The nodes corresponding to $(y_1, \ldots, y_k)$ have initially zero reliability.
6. During the BP-algorithm, the CLID algorithm of [6, 2] is applied. In every ℓ-th round of the iteration the intermediate bit y_{i_ℓ} with the smallest reliability is marked, included in the payload, and its log-likelihood is set to $+\infty$ or $-\infty$ depending on whether its value is 0 or 1.

[1] The choice of $\Omega(x)$ is crucial for performance and its specific expression is given at the end of the section.

[2] For example, $\mathbf{G}$ can be constructed row-by-row, such that every row is sampled from $\Omega(x)$ subject to the constraint that all the rows created so far are linearly independent.

The combined BP-decoding with iterative doping is continued until the intermediate bits satisfy all the parity check equations of the LT code.

7. The payload output by the compressor consists of the bits $(x_{k+1}, \ldots, x_{k+m})$, followed by the values of the doped intermediate bits.

The choice of the number of parity checks m is driven by the source entropy $H(S)$. For not too-small blocklength k, it can be observed that the CLID algorithm yields a small number of doped symbols to converge if $m/k \geq H(S) + \delta$, for some rate margin $\delta > 0$ that generally depends on the LT code, on the source statistics and on the blocklength k, while it yields a very large number of doped symbols if the coding rate m/k is too close or above the source entropy. This is due to the fact that, as one may expect, the CLID algorithm is just a "perturbation" of the BP decoder to force it to recover the source sequence with probability 1, but is not a good encoding strategy in itself. Therefore, in order to have good compression performance we should choose $m = k(H(S) + \delta)$. The longer m is, the lower the number of required doped bits, and the higher the resilience against channel error and/or erasures. Thus a tradeoff of average vs variance in redundancy is possible by tuning the degree of backoff from entropy. If the entropy of the source is now known, we compute m based on the empirical entropy of the individual source sequence estimated by the universal source modeler.

The compressor performance, and in particular the output length variance, can be improved by trying several random realizations of the LT code for a given source sequence and choose the one for which the number of doped symbols is smallest. Using LT codes makes this very simple in practice. In fact, any encoding matrix realization can be identified by a single integer number, seed of a pseudo-random number generator, that can be communicated to the decompressor at the cost of a constant (i.e., independent of the blocklength k) number of additional bits.

The decompressor also proceeds in several steps which closely mimic the compression steps using the closed-loop iterative doping algorithm:

1. Using $(x_{k+1}, \ldots, x_{k+m})$ and the marked intermediate symbols y_{i_ℓ}, all the intermediate bits $(y_1, \ldots, y_k)$ are reconstructed using a mirror image of the iterations of the BP algorithm used at the compressor.
2. Applying the encoder for the raptor code to the intermediate bits $(y_1, \ldots, y_k)$ the bits $(x_1, \ldots, x_k)$ are obtained.
3. An inverse block sorting transform recovers the original data sequence.

In instances where the data compression algorithm is run in a channel with a low rate of erasures it is still possible to run the CLID algorithm using the Quenched Belief Propagation algorithm explained in [16]. In general, when the rate of erasures is not low it is preferable not to use CLID and use instead a standard Raptor code for compression together with a BP decompressor that takes into account the probability of the data.

The choice of the output distribution $\Omega(x)$ is crucial for the performance of the algorithm. The experiments reported in this paper are all based on the following degree distribution:

$$\begin{aligned}\Omega(x) \quad = \quad & 0.008x + 0.494x^2 + 0.166x^3 + 0.073x^4 + 0.083x^5 + \\ & 0.056x^8 + 0.037x^9 + 0.056x^{19} + 0.025x^{65} + 0.003x^{66}.\end{aligned}$$

This distribution was optimized for the binary erasure channel [13], but it produces surprisingly good results for binary symmetric channels [17, 18].

V. Modelling and Universality

In this section we consider a universal version of our data compression scheme based on modelling the source statistical dependencies as a tree.

Suppose that the source sequence $x = (x_1, \ldots, x_k)$ takes values on a given q-ary alphabet $\mathcal{A}$ and is generated by a stationary ergodic tree source with S states. Then, the output of the BWT, denoted in the following by $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_k)$ converges, as $k \to \infty$, to a piecewise i.i.d. sequence [19]. This means that there exist $S' + 1$ *transition points*, denoted by $1 = t_1 < t_2 < \cdots < t_{S'} < t_{S'+1} = k + 1$ that partition $\widetilde{x}$ into segments $(\widetilde{x}_{t_j}, \ldots, \widetilde{x}_{t_{j+1}-1})$ for $j = 1, \ldots, S'$. The empirical *marginal* probability distribution of symbols in segment j is given by $\widehat{P}_j(a) = N_j(a)/(t_{j+1} - t_j)$, where we define the segment symbol counts

$$N_j(a) = \sum_{i=t_j}^{t_{j+1}-1} 1\{x_i = a\} \tag{4}$$

The number of segments S' may differ from the number of states S of the tree source but, for sufficiently large k, we have that $S' = S$ and that $\widehat{P}_j(a)$ is close to the true tree source probability distribution conditioned on state j.

Our approach consists of modelling the source tree structure and estimating its state probability by processing the BWT output sequence. Since the compressor acts on $\widetilde{x}$ by treating it as a piecewise i.i.d. sequence with given marginal probabilities, our goal is to find the most efficient piecewise i.i.d. description of $\widetilde{x}$. Generally speaking, a source statistics model $\mathcal{M}$ is given by the number of states (or segments) S', by the distinct transition points $(t_2, \ldots, t_{S'})$ and by the model segment distributions $\{Q_j(a) : j = 1, \ldots, S'\}$. The cost of a model $\mathcal{M}$ to represent $\widetilde{x}$ is measured by the total number of bits needed to describe $\widetilde{x}$, given by

$$c(\widetilde{x}, \mathcal{M}) = S'(\log_2 k + (\mathsf{q}-1)b) + \sum_{j=1}^{S'} \sum_{a \in \mathcal{A}} N_j(a) \log_2 \frac{1}{Q_j(a)} \tag{5}$$

where we spend $\log_2 k$ bits to encode the transition points[3], $(\mathsf{q}-1)b$ bits to encode each nominal segment distribution (see later for details), and

$$\sum_{a \in \mathcal{A}} N_j(a) \log_2 \frac{1}{Q_j(a)}$$

bits to encode the jth segment symbols. Notice that this length is an *estimate* of the output length of a Shannon code for encoding the symbols in segment j using the model distribution $Q_j(a)$. We take this as an estimate of the cost incurred by the fountain code compressor.

In order to find the most efficient piecewise i.i.d. source model, we follow the segment merging procedure explained in [21] with a different segment cost function. The BWT output and the original source sequence are related by

[3] We have $S' - 1$ transition points plus $\log_2 k$ bits to encode the *BWT index*, necessary to perform inverse BWT at the decompressor.

a data-dependent permutation π, such that $x_{\pi(i)} = \widetilde{x}_i$. Hence, the depth-$d$ context of each symbol $\widetilde{x}_i$ is obtained as $(x_{\pi(i)-d}, \ldots, x_{\pi(i)-1})$. By exploiting this fact, it is possible to partition $\widetilde{x}$ into segments of symbols with common context for a certain maximum depth $d_{\max}$, that is a design parameter of the algorithm. We identify segments by their context. Hence, the depth-$d_{\max}$ segments are arranged as leaves of a q-ary tree where the root is the whole sequence $\widetilde{x}$ (segment of depth 0) and where, for $0 < d < d_{\max}$, the segment with context $\mathbf{s}_1^d = (s_d, \ldots, s_1)$ has at most q children segments with contexts $(a, \mathbf{s}_1^d)$, for $a \in \mathcal{A}$.

Let the empirical *marginal* probability distribution of symbols in segment $\mathbf{s}_1^d$ be denoted by $\widehat{P}(a|\mathbf{s}_1^d) = N(a|\mathbf{s}_1^d)/L(\mathbf{s}_1^d)$, where we define the segment symbol counts

$$N(a|\mathbf{s}_1^d) = \sum_{i \in \text{ segment } \mathbf{s}_1^d} 1\{x_i = a\} \tag{6}$$

and where $L(\mathbf{s}_1^d)$ is the segment length. The cost of directly encoding segment $\mathbf{s}_1^d$ is given by

$$c\left(\mathbf{s}_1^d\right) = \log_2 k + (\mathsf{q}-1)b + \sum_{a \in \mathcal{A}} N(a|\mathbf{s}_1^d) \log_2 \frac{1}{Q(a|\mathbf{s}_1^d)} \tag{7}$$

where $Q(a|\mathbf{s}_1^d)$ is a quantized version of $\widehat{P}(a|\mathbf{s}_1^d)$ using $(\mathsf{q}-1)b$ bits. The segment merging algorithm is initialized by associating to each depth-$d_{\max}$ segment its cost $c\left(\mathbf{s}_1^{d_{\max}}\right)$. Then, for depth $d = d_{\max} - 1$ to $d = 0$, the cost associated to segment $\mathbf{s}_1^d$ is given by the minimum between the sum of the costs of its children segments and the cost of representing it directly. The children segments are merged and their corresponding branches in the segment tree are pruned if

$$\sum_{a \in \mathcal{A}} c\left(a, \mathbf{s}_1^d\right) > c\left(\mathbf{s}_1^d\right). \tag{8}$$

Otherwise, the branches are kept in the tree.

The algorithm terminates when it is not possible to prune the tree further. The leaves of the pruned tree correspond to the segments of the optimal piecewise i.i.d. model $\mathcal{M}$ for $\widehat{x}$ (subject to the tree source assumption of the original source).

As envisioned in [6] (and implemented in state-of-the-art BWT-based compressors) it is convenient to pre-process the BWT output with the move-to-front algorithm when the source has a large number of states S relatively to the blocklength k, as is often the case in practice. The move-to-front transformation replaces each symbol $\widetilde{x}_i = a$ with the number of distinct symbols appeared in $\widetilde{x}$ since the last appearance of a. Since the symbols at the input of the move-to-front algorithm are independent (or weakly dependent) the most probable symbol in the transformed sequence is 0, the next most probable is 1 and so on. The beneficial effect of move-to-front on the segment merging algorithm stems from the fact that after move-to-front the empirical distributions of the segments tend to be more similar since move-to-front implicitly implements a symbol permutation that arranges probabilities in decreasing order. Therefore, the merging algorithm is more likely to merge segments after the move-to-front operation. If the number of states is small relative to the blocklength, then the segments are long enough and it is preferable not to merge segments even if their distributions are similar. Thus in those cases, move-to-front may actually incur in a small performance degradation.

VI. Experiments

In order to compare the universal version of our scheme with the leading data compression methods such as `gzip`, `bzip` and `PPM` we will use synthetic Markov sources whose entropy is easily computable so that we can gauge how far the various methods are from the Shannon limit. We are particularly interested in the regime of moderate length. Not only the gaps from the Shannon limit would vanish (in terms of difference between rate and entropy) for long lengths, but as we mentioned above the new proposed methods are particularly useful for adaptation to source-channel schemes in data transmission applications where relatively short data packets are of interest.

Instead of experimenting with a given Markov source, we generate an ensemble of binary Markov sources whose transition probabilities are chosen at random. The number of states is equally likely to be 1, 2, 4, 8, 16, 32, and 64. A non-ergodic source ensemble is obtained by generating independently the memory length (binary logarithm of the number of states), and then conditional distributions are also generated randomly producing sources with entropy ranging from 0.05 to 0.75 bit/symbol. The same ensemble was used to test the LDPC-based scheme in [8]. Figure 1 shows a histogram of the absolute redundancy of the proposed method with `PPM`, `gzip` (Lempel-Ziv) and `bzip` (BWT-based) in the case that the source realization has 10,000 bits. None of the four methods have any prior knowledge about the source. We can see a clear advantage with respect to `gzip` and `bzip` in both variability and average redundancy, while the comparison with respect to PPM is rather competitive.

In Figure 2 a random Markov ensemble of memory length up to 4 is tested with source realizations containing 3,000 bits. All four methods suffer degradation due to the shorter blocklength (notice the x-axis scale is wider in Figure 2) but the competitive advantage of the new method is enhanced. For the sake of clarity in the figures, we do not include the results of the LDPC-based codes in [8] using the same universal modeler as in the fountain-code based algorithms. While the new algorithm offers a slight but noticeable advantage in terms of compression efficiency with respect to the LDPC-based method, the fountain-code method is much easier to implement in a universal setting as it avoids having to store a library of block codes with different rates. Thus, it can be viewed as providing a natural way to puncture a single code.

References

[1] I. Csiszar and J. Korner, *Information Theory: Coding Theorems for Discrete Memoryless Systems*, Academic, New York, 1981.

[2] G. Caire, S. Shamai, and S. Verdú, "Noiseless data compression with low density parity check codes," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, P. Gupta and G. Kramer, Eds. American Mathematical Society, 2004.

[3] P. E. Allard and A. W. Bridgewater, "A source encoding technique using algebraic codes," *Proc. 1972 Canadian Computer Conference*, pp. 201–213, June 1972.

[4] H. Ohnsorge, "Data compression system for the transmission of digitalized signals," *Conf Rec IEEE Int. Conf. on Communications*, vol. II, pp. 485–488, June 1973.

[5] T. Ancheta, "Syndrome source coding and it universal generalization," *IEEE Trans. Information Theory*, vol. 22, no. 4, pp. 432 – 436, July 1976.

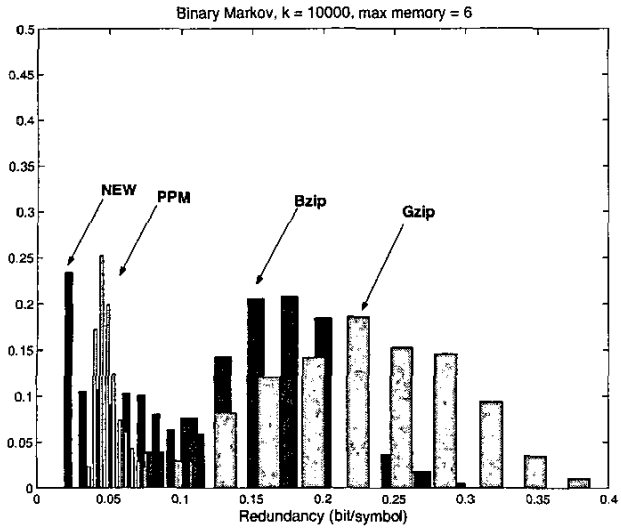

Figure 1: Nonergodic binary source; 10,000 bits

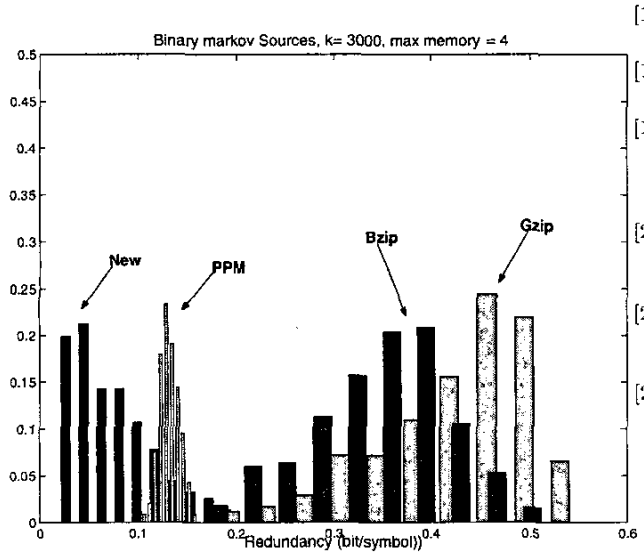

Figure 2: Nonergodic binary source; 3,000 bits

[6] G. Caire, S. Shamai, and S. Verdú, "A new data compression algorithm for sources with memory based on error correcting codes," *2003 IEEE Workshop on Information Theory*, pp. 291–295, Mar. 30- Apr. 4, 2003.

[7] G. Caire, S. Shamai, and S. Verdú, "Lossless data compression with error correction codes," *2003 IEEE Int. Symp. on Information Theory*, p. 22, June 29- July 4, 2003.

[8] G. Caire, S. Shamai, and S. Verdú, "Universal data compression with ldpc codes," *Third International Symposium On Turbo Codes and Related Topics*, pp. 55–58, Brest, France, September 1-5, 2003.

[9] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep. SRC 124, May 1994.

[10] G. Caire, S. Shamai, A. Shokrollahi and S. Verdú, "Fountain Codes for Lossless Data Compression," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, A. Barg and A. Ashkhimin, Eds. American Mathematical Society, 2005.

[11] M. Luby, "LT-codes," in *Proceedings of the 43rd Annual IEEE Symposium on the Foundations of Computer Science (STOC)*, 2002, pp. 271–280.

[12] M. G. Luby, "LT codes," *Proc. 43rd IEEE Symp. Foundations of Computer Science*, pp. 271–280, 2002.

[13] A. Shokrollahi, "Raptor codes," Preprint, 2003.

[14] E. Ordentlich, G. Seroussi, S. Verdú, K. Viswanathanand M. Weinberger, and T. Weissman, "Channel decoding of systematically encoded unknown redundant sources," in *2004 Proc. IEEE Symposium on Information Theory*, Chicago, IL, 2004.

[15] J. Hagenauer, A. Barros, and A. Schaefer, "Lossless turbo source coding with decremental redundancy," *Proc. Fifth Int. ITG Conference on Source and Channel Coding*, pp. 333–339, Jan. 2004.

[16] G. Caire, S. Shamai, and S. Verdú, "Almost-noiseless joint source-channel coding-decoding of sources with memory," *5th International ITG Conference on Source and Channel Coding (SCC)*, Jan 14-16, 2004.

[17] O. Etesami, M. Molkaraie, and A. Shokrollahi, "Raptor codes on symmetric channels," Preprint, 2003.

[18] R. Palanki and J. Yedidia, "Rateless codes on noiseless channels," *preprint*, Oct. 2003.

[19] K. Visweswariah, S. Kulkarni, and S. Verdú, "Output distribution of the Burrows-Wheeler transform," *Proc. 2000 IEEE International Symposium on Information Theory*, p. 53, June 27- July 1, 2000.

[20] N. J. Larsson, "The context trees of block sorting compression," *Proc. 1998 Data Compression Conference*, pp. 189–198, Mar. 1998.

[21] D. Baron and Y. Bresler, "An O(N) semi-predictive universal encoder via the BWT," *IEEE Trans. Information Theory*, pp. 928–937, May 2004.

[22] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Information Theory*, vol. 47, pp. 569–584, Feb. 2001.

Acknowledgement

The authors would like to thank Bertrand Ndzana Ndzana for his help with the simulations reported in this paper.