The context-tree weighting method: basic properties

The Context-Tree Weighting Method: Basic Properties

Frans M. J. Willems, *Member, IEEE*, Yuri M. Shtarkov, and Tjalling J. Tjalkens, *Member, IEEE*

Abstract—**We describe a sequential universal data compression procedure for binary tree sources that performs the "double mixture." Using a context tree, this method weights in an efficient recursive way the coding distributions corresponding to all bounded memory tree sources, and achieves a desirable coding distribution for tree sources with an unknown model and unknown parameters. Computational and storage complexity of the proposed procedure are both linear in the source sequence length. We derive a natural upper bound on the cumulative redundancy of our method for individual sequences. The three terms in this bound can be identified as coding, parameter, and model redundancy. The bound holds for all source sequence lengths, not only for asymptotically large lengths. The analysis that leads to this bound is based on standard techniques and turns out to be extremely simple. Our upper bound on the redundancy shows that the proposed context-tree weighting procedure is optimal in the sense that it achieves the Rissanen (1984) lower bound.**

Index Terms— **Sequential data compression, universal source coding, tree sources, modeling procedure, context tree, arithmetic coding, cumulative redundancy bounds.**

I. Introduction—Concepts

A *finite memory tree source* has the property that the next-symbol probabilities depend on a finite number of most recent symbols. This number in general depends on the actual values of these most recent symbols. Binary sequential universal source coding procedures for finite memory tree sources often make use of a context tree which contains for each string (context) the number of zeros and the number of ones that have followed this context, in the source sequence seen so far. The standard approach (see e.g., Rissanen and Langdon [12], Rissanen [8], [10], and Weinberger, Lempel, and Ziv [19]) is that, given the past source symbols, one uses this context tree to estimate the actual "state" of the finite memory tree source. Subsequently, this state is used to estimate the distribution that generates the next source symbol. This estimated distribution can be used in arithmetic coding procedures (see, e.g., Rissanen and Langdon [12]) to encode (and decode) the next source symbol efficiently, i.e., with low complexity and with negligible additional redundancy. After Rissanen's pioneering work in [8], Weinberger, Lempel, and Ziv [19] developed a procedure that achieves optimal exponential decay of the error probability in estimating the current state of the tree source. These authors were also able to demonstrate that their coding procedure achieves asymptotically the lower bound on the average redundancy, as stated by Rissanen ([9, Theorem 1], or [10, Theorem 1]). Recently, Weinberger, Rissanen, and Feder [21] could prove the optimality, in the sense of achieving Rissanen's lower bound on the redundancy, of an algorithm similar to that of Rissanen in [8].

An unpleasant fact about the standard approach is that one has to specify parameters (α and β in Rissanen's procedure [8] or K for the Weinberger, Lempel, and Ziv [19] method), that do not affect the asymptotic performance of the procedure, but may have a big influence on the behavior for finite (and realistic) source sequence lengths. These artificial parameters are necessary to regulate the state estimation characteristics. This gave the authors the idea that the state estimation concept may not be as natural as one believes. A better starting principle would be, just to find a good coding distribution. This more or less trivial guideline immediately suggests the application of *model weighting techniques*. An advantage of weighting procedures is that they perform well not only on the average but for each individual sequence. Model weighting (twice-universal coding) is not new. It was first suggested by Ryabko [13] for the class of finite-order Markov sources (see also [14] for a similar approach to prediction). The known literature on model weighting resulted, however, in probability assignments that require complicated sequential updating procedures. Instead of finding implementable coding methods one concentrated on achieving low redundancies. In what follows we will describe a probability assignment for *bounded memory tree sources* that allows efficient updating. This procedure, which is based on tree-recursive model-weighting, results in a coding method that is very easy to analyze, and that has a desirable performance, both in realized redundancy and in complexity.

Manuscript received August 20, 1993; revised September 1994. The material in this paper was presented in part at the IEEE Internationl Symposium on Information Theory, San Antonio, TX, January 17–22, 1993.

F. M. J. Willems and T. J. Tjalkens are with the Eindhoven University of Technology, Electrical Engineering Department, 5600 MB Eindhoven, The Netherlands.

Y. M. Shtarkov was with the Eindhoven University of Technology, Electrical Engineering Department, 5600 MB Eindhoven, The Netherlands, on leave from the Institute for Problems of Information Transmission, 101447, Moscow, GSP-4, Russia.

IEEE Log Number 9410408.

II. Binary Bounded Memory Tree Sources

A. Strings

A string s is a concatenation of binary symbols, hence $s = q_{1-l}q_{2-l}\cdots q_0$ with $q_{-i} \in \{0,1\}$, for $i = 0, 1, \cdots, l-1$. Note that we index the symbols in the string from right to left, starting with 0 and going negative. For the length of a string s

we write $l(s)$. A semi-infinite string $s = \cdots q_{-1}q_0$ has length $l(s) = \infty$. The empty string λ has length $l(\lambda) = 0$.

If we have two strings

$$s' = q'_{1-l'}q'_{2-l'}\cdots q'_0$$

and

$$s = q_{1-l}q_{2-l}\cdots q_0$$

then

$$s's \triangleq q'_{1-l'}q'_{2-l'}\cdots q'_0 q_{1-l}q_{2-l}\cdots q_0$$

is the concatenation of both. If $\mathcal{V}$ is a set of strings and $q \in \{0,1\}$, then

$$\mathcal{V} \times q \triangleq \{vq : v \in \mathcal{V}\}.$$

We say that a string $s = q_{1-l}q_{2-l}\cdots q_0$ is a *suffix* of the string $s' = q'_{1-l'}q'_{2-l'}\cdots q'_0$, if $l \le l'$ and $q_{-i} = q'_{-i}$ for $i = 0..l-1$. The empty string λ is a suffix of all strings.

B. Binary Bounded Memory Tree Source Definition

A *binary tree source* generates a sequence $x_{-\infty}^{\infty}$ of digits assuming values in the alphabet $\{0,1\}$. We denote by x_m^n the sequence $x_m x_{m+1}\cdots x_n$, and allow m and n to be infinitely large. For $n < m$ the sequence x_m^n is empty, denoted by ϕ.

The statistical behavior of a binary *finite memory* tree source can be described by means of a *suffix set* $\mathcal{S}$. This suffix set is a collection of binary strings $s(k)$, with $k = 1, 2, \cdots, |\mathcal{S}|$. We require it to be *proper* and *complete*. Properness of the suffix set implies that no string in $\mathcal{S}$ is a *suffix* of any other string in $\mathcal{S}$. Completeness guarantees that each semi-infinite sequence (string) $\cdots x_{n-2}x_{n-1}x_n$ has a suffix that belongs to $\mathcal{S}$. This suffix is unique since $\mathcal{S}$ is proper.

Let $D \in \{0.1.\cdots\}$ be fixed throughout this paper. A *bounded memory* tree source has a suffix set $\mathcal{S}$ that satisfies $l(s) \le D$ for all $s \in \mathcal{S}$. We say that the source has memory not larger than D.

The properness and completeness of the suffix set make it possible to define the *suffix function* $\beta_{\mathcal{S}}(\cdot)$. This function maps semi-infinite sequences onto their unique suffix s in $\mathcal{S}$. Since all suffixes in $\mathcal{S}$ have length not larger than D, only the last D symbols of a semi-infinite sequence determine its suffix in $\mathcal{S}$. To each suffix s in $\mathcal{S}$ there corresponds a *parameter* θ_s. Each parameter (i.e., the probability of a source symbol being 1) assumes a value in $[0,1]$ and specifies a probability distribution over $\{0,1\}$. Together, all parameters form the parameter vector

$$\Theta_{\mathcal{S}} \triangleq \{\theta_s : s \in \mathcal{S}\}.$$

If the tree source has emitted the semi-infinite sequence $x_{-\infty}^{t-1}$ up to now, the suffix function tells us that the parameter for generating the next binary digit x_t of the source is θ_s, where $s = \beta_{\mathcal{S}}(x_{t-D}^{t-1})$. Thus

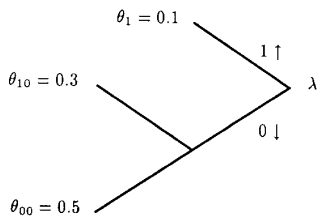

Fig. 1. Model (suffix set) and parameters.

Definition 1: The *actual* next-symbol probabilities for a bounded memory tree source with suffix set $\mathcal{S}$ and parameter vector $\Theta_{\mathcal{S}}$ are

$$P_a(X_t = 1|x_{t-D}^{t-1}, \mathcal{S}, \Theta_{\mathcal{S}}) = 1 - P_a(X_t = 0|x_{t-D}^{t-1}, \mathcal{S}, \Theta_{\mathcal{S}}) \triangleq \theta_{\beta_{\mathcal{S}}(x_{t-D}^{t-1})}. \quad \text{for all } t. \tag{1}$$

The actual block probabilities are now products of actual next-symbol probabilities, i.e.

$$P_a(X_1^t = x_1^t|x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}}) = \prod_{\tau=1}^{t} P_a(X_\tau = x_\tau|x_{\tau-D}^{\tau-1}, \mathcal{S}, \Theta_{\mathcal{S}}).$$

All tree sources with the same suffix set are said to have the same *model*. Model and suffix set are equivalent. The set of all tree models having memory not larger than D is called the *model class* $\mathcal{C}_D$. It is possible to specify a model in this model class by a *natural* code by encoding the suffix set $\mathcal{S}$ recursively. The code of $\mathcal{S}$ is the code of the empty string λ. The code of a string s is void if $l(s) = D$; otherwise, it is 0 if $s \in \mathcal{S}$ and 1 followed by the codes of the strings $0s$ and $1s$ if $s \notin \mathcal{S}$. If we use this natural code, the number of bits that are needed to specify a model $\mathcal{S} \in \mathcal{C}_D$ is equal to $\Gamma_D(\mathcal{S})$, where

Definition 2: $\Gamma_D(\mathcal{S})$, the cost of a model $\mathcal{S}$ with respect to model class $\mathcal{C}_D$ is defined as

$$\Gamma_D(\mathcal{S}) \triangleq |\mathcal{S}| - 1 + |\{s : s \in \mathcal{S}, l(s) \ne D\}| \tag{2}$$

where it is assumed that $\mathcal{S} \in \mathcal{C}_D$.

Example 1: Let $D = 3$. Consider a source with suffix set $\mathcal{S} = \{00, 10, 1\}$ and parameters $\theta_{00} = 0.5$, $\theta_{10} = 0.3$, and $\theta_1 = 0.1$ (see Fig. 1). The (conditional) probability of the source generating the sequence 0110100 given the past symbols $\cdots 010$ can be calculated as follows:

$$P_a(0110100|\cdots 010) = (1-\theta_{10})\theta_{00}\theta_1(1-\theta_1)\theta_{10}(1-\theta_1)(1-\theta_{10}) = 0.0059535. \tag{3}$$

Since $D = 3$, the model (suffix set) $\mathcal{S}$ can be specified by

$$\text{code}(\mathcal{S}) = \text{code}(\lambda) = 1 \;\; \text{code}(0) \;\; \text{code}(1) = 1 \;\; 1 \;\; \text{code}(00) \;\; \text{code}(10) \;\; 0 = 1 \;\; 1 \;\; 0 \;\; 0 \;\; 0. \tag{4}$$

Tree sources are related to FSMX sources that were first described by Rissanen [10]. FSMX sources can be considered as tree sources whose suffix set $\mathcal{S}$ is *closed*. A suffix set is said to be closed if the *generator* of each suffix $s \in \mathcal{S}$ belongs to $\mathcal{S}$ or is a suffix of an $s \in \mathcal{S}$. The generator of a suffix $s = q_{1-l}\cdots q_{-1}q_0$ is $q_{1-l}\cdots q_{-1}$. Note that $\mathcal{S} = \{00.010.110.1\}$ is a tree model, but not an FSMX model.

Each finite memory tree source with suffix set $\mathcal{S}$ has a finite-state machine implementation. The number of states is then $|\mathcal{S}|$ or more. Only for tree sources with a closed suffix set $\mathcal{S}$ (i.e., for FSMX sources) the number of states is equal to $|\mathcal{S}|$.

III. Codes and Redundancy

Let $T \in \{1, 2, \cdots\}$. Instead of the source sequence $x_1^T \triangleq x_1 x_2 \cdots x_T$ itself, the *encoder* sends a codeword $c^L \triangleq c_1 c_2 \cdots c_L$ consisting of $\{0,1\}$-components to the *decoder*. The decoder must be able to reconstruct the source sequence x_1^T from this codeword.

We assume that both the encoder and the decoder have access to the past source symbols $x_{1-D}^0 = x_{1-D} \cdots x_{-1} x_0$, so that implicitely the suffix that determines the probability distribution of the first source symbols, is available to them. A codeword that is formed by the encoder therefore depends not only on the source sequence x_1^T but also on x_{1-D}^0. To denote this functional relationship we write $c^L(x_1^T | x_{1-D}^0)$. The length of the codeword, in binary digits, is denoted as $L(x_1^T | x_{1-D}^0)$.

We restrict ourselves to *prefix codes* here (see [3, ch. 5]). These codes are not only uniquely decodable but also *instantaneous* or *self-punctuating* which implies that you can immediately recognize a codeword when you see it. The set of codewords that can be produced for a fixed x_{1-D}^0 form a prefix code, i.e., no codeword is the prefix of any other codeword in this set. All sequences $c_1 c_2 \cdots c_l$, for some $l = 1, 2, \cdots, L$ are a prefix of $c_1 c_2 \cdots c_L$.

The codeword lengths $L(x_1^T | x_{1-D}^0)$ determine the *individual redundancies*.

Definition 3: The individual redundancy

$$\rho(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}})$$

of a sequence x_1^T given the past symbols x_{1-D}^0, with respect to a source with model $\mathcal{S} \in \mathcal{C}_D$ and parameter vector $\Theta_{\mathcal{S}}$, is defined as[1]

$$\rho(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}}) \triangleq L(x_1^T | x_{1-D}^0) - \log \frac{1}{P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}})} \quad (5)$$

where $L(x_1^T | x_{1-D}^0)$ is the length of the codeword that corresponds to x_1^T given x_{1-D}^0. We consider only sequences x_1^T with positive probability $P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}})$.

The value

$$\log (1 / P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_{\mathcal{S}}))$$

can be regarded as the *information* contained in x_1^T given the past x_{1-D}^0. It is often called the *ideal codeword length*. Note that we do not divide the redundancies by the source sequence length T, we consider only *cumulative* redundancies. Note also that our redundancies can be negative.

The objective in universal source coding is to design methods that achieve small individual redundancies with respect to all sources in a given class. Since it is also very important that these methods have low (storage and computational) complexity, it would be more appropriate to say that the emphasis in source coding is on finding a desirable tradeoff between achieving small redundancies and keeping the complexity low.

[1] The basis of the $\log(\cdot)$ is assumed to be 2, throughout this paper.

IV. Arithmetic Coding

An *arithmetic* encoder *computes* the codeword that corresponds to the actual source sequence. The corresponding decoder reconstructs the actual source sequence from this codeword again by computation. Using arithmetic codes it is possible to process source sequences with a large length T. This is often needed to reduce the redundancy per source symbol.

Arithmetic codes are based on the Elias algorithm (unpublished, but described by Abramson [1] and Jelinek [4]) or on enumeration (e.g., Schalkwijk [15] and Cover [2]). Arithmetic coding became feasable only after Rissanen [7] and Pasco [6] had solved the accuracy issues that were involved. We will not discuss such issues here. Instead, we will assume that all computations are carried out with *infinite precision*.

Suppose that the encoder and decoder both have access to, what is called the *coding distribution*

$$P_c(x_1^t), x_1^t \in \{0,1\}^t, t = 0, 1, \cdots, T.$$

We require that this distibution satisfies

$$P_c(\phi) = 1,$$
$$P_c(x_1^{t-1}) = P_c(x_1^{t-1}, X_t = 0) + P_c(x_1^{t-1}, X_t = 1), \quad \text{for all } x_1^{t-1} \in \{0,1\}^{t-1}, t = 1, \cdots, T$$

and

$$P_c(x_1^T) > 0, \qquad \text{for all } \textit{possible } x_1^T \in \{0,1\}^T \quad (6)$$

where possible sequences are sequences that can actually occur, i.e., sequences x_1^T with $P_a(x_1^T) > 0$. Note that ϕ stands for the empty sequence (x_1^0).

In Appendix I we describe the Elias algorithm. It results in the following theorem.

Theorem 1: Given a coding distribution

$$P_c(x_1^t), x_1^t \in \{0,1\}^t, t = 0, 1, \cdots, T$$

the Elias algorithm achieves codeword lengths $L(x_1^T)$ that satisfy

$$L(x_1^T) < \log \frac{1}{P_c(x_1^T)} + 2 \quad (7)$$

for all possible $x_1^T \in \{0,1\}^T$. The codewords form a prefix code.

The difference between the codeword length $L(x_1^T)$ and $\log(1/P_c(x_1^T))$ is always less than 2 bits. We say that the individual *coding redundancy* is less than 2 bits.

We conclude this section with the observation that the Elias algorithm combines an acceptable coding redundancy with a desirable *sequential* implementation. The number of operations is linear in the source sequence length T. It is crucial, however, that the encoder and decoder have access to the probabilities $P_c(x_1^{t-1}, X_t = 0)$ and $P_c(x_1^{t-1}, X_t = 1)$ after having processed $x_1 x_2 \cdots x_{t-1}$. If this is the case we say that the coding distribution is *sequentially available*.

It should be noted that our view of an arithmetic code is slightly different from the usual. We assume that block probabilities are fed into the encoder and decoder and not conditional probabilities as usual. The reason for this is that it creates a better match between our modeling algorithm and the arithmetic code, and avoids multiplications.

If we are ready to accept a loss of at most 2 bits coding redundancy, we are now left with the problem of finding good, sequentially available, coding distributions.

V. Probability Estimation

The probability that a memoryless source with parameter θ generates a sequence with a zeros and b ones is $(1-\theta)^a\theta^b$. If we *weight* this probability over all θ with a $(\frac{1}{2},\frac{1}{2})$-Dirichlet distribution we obtain the so-called Krichevsky–Trofimov estimate (see [5]).

Definition 4: The Krichevski–Trofimov (KT) estimated probability for a sequence containing $a \geq 0$ zeros and $b \geq 0$ ones is defined as

$$P_e(a,b) \triangleq \int_0^1 \frac{1}{\pi\sqrt{(1-\theta)\theta}}(1-\theta)^a\theta^b\,d\theta. \tag{8}$$

This estimator has properties that are listed in the lemma that follows. The lemma is proved in Appendix II.

Lemma 1: The KT-probability estimator $P_e(a,b)$

1) can be computed sequentially, i.e., $P_e(0,0) = 1$, and for $a \geq 0$ and $b \geq 0$

$$P_e(a+1,b) = \frac{a+\frac{1}{2}}{a+b+1}\cdot P_e(a,b)$$

and

$$P_e(a,b+1) = \frac{b+\frac{1}{2}}{a+b+1}\cdot P_e(a,b) \tag{9}$$

2) satisfies, for $a+b \geq 1$, the following inequality:

$$P_e(a,b) \geq \frac{1}{2}\cdot\frac{1}{\sqrt{a+b}}\left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b. \tag{10}$$

The sequential behavior of the KT-estimator was studied by Shtarkov [17]. Another estimator, the Laplace estimator, was investigated by Rissanen [10], [11]. This estimator can be obtained by weighting $(1-\theta)^a\theta^b$ with θ uniform over $[0,1]$.

For the KT-estimator the *parameter redundancy* can be uniformly bounded, using the lower bound (see Lemma 1) on $P_e(a,b)$, i.e.

$$\log\frac{(1-\theta)^a\theta^b}{P_e(a,b)} \leq \log\frac{(1-\theta)^a\theta^b}{\frac{1}{2}\frac{1}{\sqrt{a+b}}\left(\frac{a}{a+b}\right)^a\left(\frac{b}{a+b}\right)^b} \leq \frac{1}{2}\log(a+b)+1 \tag{11}$$

for all $a+b \geq 1$ and all $\theta \in [0,1]$. It is impossible to prove such a uniform bound for the Laplace estimator.

VI. Coding for an Unknown Tree Source

A. Definition of the Context-Tree Weighting Method

Consider the case where we have to compress a sequence which is (supposed to be) generated by a tree source, whose suffix set $\mathcal{S} \in \mathcal{C}_D$ and parameter vector $\Theta_{\mathcal{S}}$ are unknown to the encoder and the decoder. We will define a *weighted* coding distribution for this situation, study its performance, and discuss its implementation. The coding distribution is based on the concept of a *context tree* (see Fig. 2).

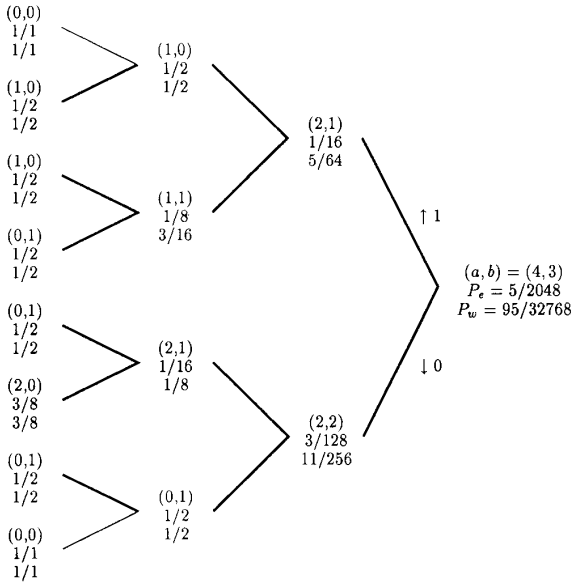

Fig. 2. Weighted context tree $\mathcal{T}_3$ for $h_1^7 = 0110100$ and $x_{1-D}^0 = \cdots 010$.

Definition 5: The context tree $\mathcal{T}_D$ is a set of *nodes* labeled s, where s is a (binary) string with length $l(s)$ such that $0 \leq l(s) \leq D$. Each node $s \in \mathcal{T}_D$ with $l(s) < D$, "splits up" into two nodes, $0s$ and $1s$. The node s is called the *parent* of the nodes $0s$ and $1s$, who in turn are the *children* of s. To each node $s \in \mathcal{T}_D$, there correspond counts $a_s \geq 0$ and $b_s \geq 0$. For the children $0s$ and $1s$ of parent node s, the counts must satisfy $a_{0s} + a_{1s} = a_s$ and $b_{0s} + b_{1s} = b_s$.

Now, to each node there corresponds a *weighted probability*. This weighted probability is defined recursively on the context tree $\mathcal{T}_D$. Without any doubt, this is the basic definition in this paper.

Definition 6: To each node $s \in \mathcal{T}_D$, we assign a weighted probability P_w^s which is defined as

$$P_w^s \triangleq \begin{cases} \frac{1}{2}P_e(a_s,b_s) + \frac{1}{2}P_w^{0s}P_w^{1s}, & \text{for } 0 \leq l(s) < D \\ P_e(a_s,b_s). & \text{for } l(s) = D. \end{cases} \tag{12}$$

The context tree together with the weighted probabilities of the nodes is called a *weighted context tree*.

This definition shows a weighting of both the estimated probability in a node and the product of the weighted probabilities that correspond to its children. The next lemma gives another way of looking at the weighting that is performed in (12). It explains that a weighted probability of a node can be regarded as a weighting over the estimated probabilities corresponding to all (sub-) models that live above this node. The cost (see (2)) of a (sub-) model determines its weighting factor. The proof of this lemma can be found in Appendix III.

Lemma 2: The weighted probability P_w^s of a node $s \in \mathcal{T}_D$ with $l(s) = d$ for $0 \leq d \leq D$ satisfies

$$P_w^s = \sum_{\mathcal{U} \in \mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{U})} \prod_{u \in \mathcal{U}} P_e(a_{us}, b_{us}) \tag{13}$$

with

$$\sum_{\mathcal{U} \in \mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{U})} = 1.$$

The summation is over all complete and proper suffix sets $\mathcal{U}$.

To be able to define a weighted coding distribution, we assume that the counts (a_s, b_s), $s \in \mathcal{T}_D$ are determined by the source sequence x_1^t seen up to now, assuming that x_{1-D}^0 are the past symbols.

Definition 7: For each $s \in \mathcal{T}_D$ let $a_s(x_1^t | x_{1-D}^0)$, respectively $b_s(x_1^t | x_{1-D}^0)$, be the number of times that $x_\tau = 0$, respectively $x_\tau = 1$, in x_1^t for $1 \leq \tau \leq t$ such that $x_{\tau-l(s)}^{\tau-1} = s$. The weighted probabilities corresponding to the nodes $s \in \mathcal{T}_D$ are now denoted by $P_w^s(x_1^t | x_{1-D}^0)$. For any sequence x_{1-D}^0 of past symbols, we define our weighted coding distribution as

$$P_c(x_1^t | x_{1-D}^0) \triangleq P_w^\lambda(x_1^t | x_{1-D}^0) \tag{14}$$

for all $x_1^t \in \{0,1\}^t, t = 0, 1, \cdots, T$, where λ is the root node of the context tree $\mathcal{T}_D$.

This coding distribution determines the *context-tree weighting method.* Note that the counts indeed satisfy the restrictions mentioned in Definition 6.1. To verify that it satisfies (6) we formulate a lemma. The proof of this lemma can be found in Appendix IV.

Lemma 3: Let $t = 1, 2, \cdots, T$. If $s \in \mathcal{T}_D$ is not a suffix of x_{t-D}^{t-1}, then

$$\begin{aligned} P_w^s(x_1^{t-1}, X_t = 0 | x_{1-D}^0) &= P_w^s(x_1^{t-1}, X_t = 1 | x_{1-D}^0) \\ &= P_w^s(x_1^{t-1} | x_{1-D}^0) \end{aligned} \tag{15}$$

and, if s is a suffix of x_{t-D}^{t-1}, then

$$\begin{aligned} P_w^s(x_1^{t-1}, X_t = 0 | x_{1-D}^0) + P_w^s(x_1^{t-1}, X_t = 1 | x_{1-D}^0) \\ = P_w^s(x_1^{t-1} | x_{1-D}^0). \end{aligned} \tag{16}$$

To check that the weighted coding distribution defined in (14) is an allowable coding distribution observe that $P_w^\lambda(\phi | x_{1-D}^0) = 1$. Subsequently, note that Lemma 3 states that

$$\begin{aligned} P_w^\lambda(x_1^{t-1}, X_t = 0 | x_{1-D}^0) + P_w^\lambda(x_1^{t-1}, X_t = 1 | x_{1-D}^0) \\ = P_w^\lambda(x_1^{t-1} | x_{1-D}^0) \end{aligned}$$

since λ is a suffux of all strings x_{t-D}^{t-1}. From this we may conclude that our weighted coding distribution satisfies (6) after having verified that weighted probabilities are always positive.

We are now ready to investigate the redundancy of the context tree weighting method.

B. An Upper Bound on the Redundancy

First we give a definition.

Definition 8: Let

$$\gamma(z) \triangleq \begin{cases} z, & \text{for } 0 \leq z < 1 \\ \frac{1}{2} \log z + 1, & \text{for } z \geq 1 \end{cases} \tag{17}$$

hence $\gamma(\cdot)$ is a convex-$\cap$ continuation of $\frac{1}{2} \log z + 1$ for $0 \leq z < 1$ satisfying $\gamma(0) = 0$.

The basic result concerning the context-tree weighting technique can be stated now.

Theorem 2: The individual redundancies with respect to any source with model $\mathcal{S} \in \mathcal{C}_D$ and parameter vector $\Theta_\mathcal{S}$ are upper-bounded by

$$\rho(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S}) < \Gamma_D(\mathcal{S}) + |S| \gamma\left(\frac{T}{|\mathcal{S}|}\right) + 2 \tag{18}$$

for all $x_1^T \in \{0,1\}^T$, for any sequence of past symbols x_{1-D}^0, if we use the weighted coding distribution specified in (14).

Note that (18) can be rewritten as (see (19) at the bottom of this page). The redundancy bound in Theorem 2 holds with respect to *all* sources with model $\mathcal{S} \in \mathcal{C}_D$ and parameter vector $\Theta_\mathcal{S}$, and *not only the actual source*. Using the definition of redundancy (see (5)) we therefore immediately obtain an upper bound on the codeword lengths.

Corollary 1: Using the coding distribution in (14), the codeword lengths $L(x_1^T | x_{1-D}^0)$ are upper-bounded by

$$\begin{aligned} L(x_1^T | x_{1-D}^0) < \min_{\mathcal{S} \in \mathcal{C}_D} \Bigg(\min_{\Theta_\mathcal{S}} \log \frac{1}{P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S})} + \Gamma_D(\mathcal{S}) \\ + |S| \gamma\left(\frac{T}{|\mathcal{S}|}\right) \Bigg) + 2 \end{aligned} \tag{20}$$

for all $x_1^T \in \{0,1\}^T$, for any sequence x_{1-D}^0 of past symbols.

Proof: Consider a sequence $x_1^T \in \{0,1\}^T$, a suffix set $\mathcal{S} \in \mathcal{C}_D$, and a parameter vector $\Theta_\mathcal{S}$. Let

$$a_s = a_s(x_1^T | x_{1-D}^0)$$

and

$$b_s = b_s(x_1^T | x_{1-D}^0)$$

for all $s \in \mathcal{T}_D$. We split up the individual redundancy in three terms, model redundancy, parameter redundancy, and coding redundancy

$$\begin{aligned} \rho(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S}) &= L(x_1^T | x_{1-D}^0) \\ &\quad - \log \frac{1}{P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S})} \\ &= \log \frac{\prod_{s \in \mathcal{S}} P_e(a_s, b_s)}{P_c(x_1^T | x_{1-D}^0)} \\ &\quad + \log \frac{P_a(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S})}{\prod_{s \in \mathcal{S}} P_e(a_s, b_s)} \end{aligned}$$

$$\rho(x_1^T | x_{1-D}^0, \mathcal{S}, \Theta_\mathcal{S}) < \begin{cases} \Gamma_D(\mathcal{S}) + T + 2, & \text{for } T = 1, \cdots, |\mathcal{S}| - 1 \\ \Gamma_D(\mathcal{S}) + \frac{|\mathcal{S}|}{2} \log \frac{T}{|\mathcal{S}|} + |\mathcal{S}| + 2. & \text{for } T = |\mathcal{S}|, |\mathcal{S}| + 1, \cdots. \end{cases} \tag{19}$$

$$+\left(L(x_1^T|x_{1-D}^0)-\log\frac{1}{P_c(x_1^T|x_{1-D}^0)}\right). \tag{21}$$

For the last term, the coding redundancy, we obtain, using Theorem 1, that

$$L(x_1^T|x_{1-D}^0)-\log\frac{1}{P_c(x_1^T|x_{1-D}^0)}<2. \tag{22}$$

We treat the parameter redundancy, the middle term, as follows:

$$\begin{aligned}\log\frac{P_a(x_1^T|x_{1-D}^0,\mathcal{S},\Theta_{\mathcal{S}})}{\prod_{s\in\mathcal{S}}P_e(a_s,b_s)}&=\sum_{s\in\mathcal{S}}\log\frac{(1-\theta_s)^{a_s}\theta_s^{b_s}}{P_e(a_s,b_s)}\\&\le\sum_{s\in\mathcal{S}:a_s+b_s>0}\left(\frac{1}{2}\log(a_s+b_s)+1\right)\\&=|\mathcal{S}|\sum_{s\in\mathcal{S}}\frac{1}{|\mathcal{S}|}\gamma(a_s+b_s)\\&\le|\mathcal{S}|\gamma\left(\sum_{s\in\mathcal{S}}\frac{a_s+b_s}{|\mathcal{S}|}\right)=|\mathcal{S}|\gamma\left(\frac{T}{|\mathcal{S}|}\right).\end{aligned} \tag{23}$$

The product

$$\prod_{s\in\mathcal{S}}P_e(a_s,b_s)$$

makes it possible to split up the parameter redundancy in $|\mathcal{S}|$ terms representing the parameter redundancies corresponding to each of the $|\mathcal{S}|$ suffixes in $\mathcal{S}$. The term corresponding to suffix $s\in\mathcal{S}$ can be upper-bounded by $\frac{1}{2}\log(a_s+b_s)+1$ as we have seen before in (11); however, only for $a_s+b_s>0$. For $a_s+b_s=0$ such a term does not contribute to the redundancy. This is why we have introduced the function γ. Its $\cap$-convexity makes it possible to apply Jensen's inequality (see Cover and Thomas, [3, p. 25]).

What remains to be investigated is the first term in (21), the *model redundancy* term. It follows from Lemma 2 that

$$P_w^\lambda=\sum_{\mathcal{U}\in\mathcal{C}_D}2^{-\Gamma_D(\mathcal{U})}\prod_{s\in\mathcal{U}}P_e(a_s,b_s)\ge 2^{-\Gamma_D(\mathcal{S})}\prod_{s\in\mathcal{S}}P_e(a_s,b_s). \tag{24}$$

Using (14) we obtain the following upper bound for the model redundancy:

$$\log\frac{\prod_{s\in\mathcal{S}}P_e(a_s,b_s)}{P_c(x_1^T|x_{1-D}^0)}\le\log\frac{1}{2^{-\Gamma_D(\mathcal{S})}}=\Gamma_D(\mathcal{S}). \tag{25}$$

Combining (22), (23), and (25) in (21) yields the theorem. ■

Theorem 2 is the basic result in this paper. In this theorem we recognize beside the coding and parameter redundancy the model redundancy. Model redundancy is a consequence of not knowing the (actual, or best in the sense of minimizing (20)) model $\mathcal{S}$, and therefore not being able to take distribution

$$\prod_{s\in\mathcal{S}}P_e(a_s,b_s)$$

as coding distribution. This results in a loss, the model redundancy, which is upper-bounded by $\Gamma_D(\mathcal{S})$ bits. Note that in Section II we have described a natural code that would need $\Gamma_D(\mathcal{S})$ bits to specify the model $\mathcal{S}$. Therefore, our weighted method is at least as good as a two-pass method, in which first the best model is determined and transmitted, followed by the code for the sequence given that model.

Example 2: Suppose a source generated the sequence $x_1^T=0110100$ with sequence of past symbols $x_{1-D}^0=\cdots 010$.

For $D=3$ we have plotted the weighted context tree $\mathcal{T}_D$ in Fig. 2. Node s contains the counts (a_s,b_s), the Krichevsky–Trofimov estimate $P_e(a_s,b_s)$, and the weighted probability P_w^s. The coding probability corresponding to this sequence is $95/32768$.

The upper bound for the model redundancy with respect to the model $\mathcal{S}=\{00,10,1\}$ of the source from Example 1 is $\Gamma_D(\mathcal{S})=5$ bits. This also follows quite easily from

$$\begin{aligned}P_w^\lambda&\ge\frac{1}{2}P_w^0P_w^1\ge\frac{1}{2}\left(\frac{1}{2}P_w^{00}P_w^{10}\right)P_w^1\\&\ge\frac{1}{2}\left(\frac{1}{2}\left(\frac{1}{2}P_e(a_{00},b_{00})\right)\left(\frac{1}{2}P_e(a_{10},b_{10})\right)\right)\\&\quad\cdot\left(\frac{1}{2}P_e(a_1,b_1)\right).\end{aligned} \tag{26}$$

C. Implementation of the Context-Tree Weighting Method

Before discussing implementation issues we refer to Appendix I for notation concerning arithmetic encoding and decoding.

1) Encoding: First we set

$$B(\phi|x_{1-D}^0):=0.$$

Then, for $t=1,2,\cdots,T$, we create nodes

$$s(d):=x_{t-d}^{t-1}$$

for $d=0,1,\cdots,D$ (if necessary), we do a dummy 0-update on these nodes to find

$$P_c(x_1^{t-1},X_t=0|x_{1-D}^0)$$

we update $B(x_1^{t-1}|x_{1-D}^0)$ to

$$B(x_1^{t-1},X_t=x_t|x_{1-D}^0)$$

and we then do the actual update of the nodes $s(d)$ for $d=0,1,\cdots,D$ for $X_t=x_t$. This results in

$$P_c(x_1^{t-1},X_t=x_t|x_{1-D}^0).$$

After having processed $x_1x_2\cdots x_T$ we compute $c^L(x_1^T|x_{1-D}^0)$ from $B(x_1^T|x_{1-D}^0)$ and $P_c(x_1^T|x_{1-D}^0)$.

2) Decoding: First we set

$$B(\phi|x_{1-D}^0) := 0$$

and determine F_∞ from

$$c_1 c_2 \cdots c_{L(x_1^T)} c_{L(x_1^T)+1}, \cdots.$$

Then, for $t = 1, 2, \cdots, T$, we create nodes

$$s(d) := x_{t-d}^{t-1}$$

for $d = 0, 1, \cdots, D$ (if necessary), we do a dummy 0-update on these nodes to find

$$P_c(x_1^{t-1}, X_t = 0|x_{1-D}^0)$$

we compare F_∞ with

$$B(x_1^{t-1}|x_{1-D}^0) + P_c(x_1^{t-1}, X_t = 0|x_{1-D}^0)$$

to find x_t, update $B(x_1^{t-1}|x_{1-D}^0)$ to

$$B(x_1^{t-1}, X_t = x_t|x_{1-D}^0)$$

and we then do the actual update of the nodes $s(d)$ for $d = 0, 1, \cdots, D$ for $X_t = x_t$. This yields

$$P_c(x_1^{t-1}, X_t = x_t).$$

After having processed $x_1 x_2 \cdots x_T$ we compute $L(x_1^T)$ from $P_c(x_1^T)$ so that we know the start of the next codeword.

3) Comments: We assume that a node $s \in \mathcal{T}_D$ contains the pair (a_s, b_s), the estimated probability $P_e(a_s, b_s)$, and the weighted probability P_w^s. When a node is created, the counts a_s and b_s are made 0, the probabilities $P_e(a_s, b_s)$ and P_w^s are made 1.

Doing a dummy 0-update of the nodes $s(d)$ for $d = 0, 1, \cdots, D$ means that we assume that $X_t = 0$. Then, for $d = D, D-1, \cdots, 0$, we update as indicated by (9) in Lemma 1

$$\tilde{P}_e(a_{s(d)} + 1, b_{s(d)}) := P_e(a_{s(d)}, b_{s(d)}) \cdot \frac{a_{s(d)} + 1/2}{a_{s(d)} + b_{s(d)} + 1} \tag{27}$$

where the tilde above a variable indicates that this variable is a temporary one. After that we form

$$\tilde{P}_w^{s(D)} := \tilde{P}_e(a_{s(D)} + 1, b_{s(D)})$$

and for $d = D-1, D-2, \cdots, 0$, we compute

$$\tilde{P}_w^{s(d)} := \frac{1}{2}\tilde{P}_e(a_{s(d)} + 1, b_{s(d)}) + \frac{1}{2}\tilde{P}_w^{x_{t-d-1}s(d)} P_w^{\overline{x}_{t-d-1}s(d)} \tag{28}$$

where we note that

$$x_{t-d-1}s(d) = s(d+1)$$

so $P_w^{s(d+1)}$ was changed, and $P_w^{\overline{x}_{t-d-1}s(d)}$ has remained the same (see Lemma 3). All this eventually results in

$$P_c(x_1^{t-1}, X_t = 0|x_{1-D}^0).$$

It will be clear from (39) that (see (29) at the bottom of this page). It should be noted that we use block probabilities to feed into the arithmetic encoder and decoder instead of conditional probabilities as usual. This avoids multiplications in the arithmetic encoder and decoder, which is a pleasant side effect of the weighted approach.

If $X_t = 0$, the actual update is identical to (27) and (28), the only difference is that now we update $P_e(a_s, b_s)$ and P_w^s and increment a_s instead of computing the temporary values $\tilde{P}_e(a_s, b_s)$ and $\tilde{P}_w^s$. If $X_t = 1$, the actual update requires incrementing of b_s, etc. Note that we only have to update the nodes $s(d)$ for $d = 0, 1, \cdots, D$, the nodes along the path in the context tree that is determined by the past symbols x_{t-D}^{t-1}.

The codeword $c^L(x_1^T)$ is finally computed as in definition (36) in Appendix I and transmitted to the decoder.

The decoder forms F_∞ as in (35) in Appendix I. Note that F_∞ is compared to the threshold

$$D(x_1^{t-1}) = B(x_1^{t-1}|x_{1-D}^0) + P_c(x_1^{t-1}, X_t = 0|x_{1-D}^0)$$

see (42), Appendix I. Finally, the length $L(x_1^T)$ is computed as in Definition 11.

4) Complexity Issues: For each symbol x_t we have to visit $D+1$ nodes. Some of these nodes have to be created first. From this it follows that the total number of allocated nodes cannot be more than $T(D+1)$. This makes the *storage complexity* not more than *linear* in T. Note also that the number of nodes cannot be more than $2^{D+1} - 1$, the total number of nodes in $\mathcal{T}_D$. This shows exponential behavior in D. We did not take into account here, that for infinite precision arithmetic, the number of digits that are needed to specify the counts a_s and b_s and the probabilities $P_e(a_s, b_s)$ and P_w^s, is growing with increasing t, making the storage space for one node measured in, e.g., bytes getting bigger each time.

The *computational complexity*, i.e. the number of additions, multiplications, and divisions, is proportional to the number of nodes that are visited, which is $T(D+1)$. Therefore, this complexity is also *linear* in T. Again we have neglected the fact here, that for infinite precision arithmetic the number of digits that are needed to specify the counts a_s and b_s and the probabilities $P_e(a_s, b_s)$ and P_w^s, is growing rapidly, making additions, multiplications, and divisions becoming more complex with increasing t.

VII. Other Weightings

The coding distribution defined by (12) (and (14)) yields model cost not more than $2|\mathcal{S}| - 1$, i.e., linear in $|\mathcal{S}|$, if we assume that $\mathcal{S}$ has no leaves at depth D. This is achieved

$$B(x_1^{t-1}, X_t = x_t|x_{1-D}^0) := \begin{cases} B(x_1^{t-1}|x_{1-D}^0), & \text{if } x_t = 0 \\ B(x_1^{t-1}|x_{1-D}^0) + P_c(x_1^{t-1}, X_t = 0|x_{1-D}^0), & \text{if } x_t = 1. \end{cases} \tag{29}$$

by giving equal weight to $P_e(a_s, b_s)$ and $P_w^{0s}P_w^{1s}$ in each (internal) node $s \in \mathcal{T}_D$.

It is quite possible, however, to assume that these weights are not equal, and even to suppose that they are different for different nodes s. In this section we will assume that the weigthing in a node s depends on the depth $l(s)$ of this node in the context tree $\mathcal{T}_D$. Hence

$$P_w^s \triangleq \alpha_{l(s)}P_e(a_s, b_s) + (1 - \alpha_{l(s)})P_w^{0s}P_w^{1s}, \quad \text{with } \alpha_D = 1. \tag{30}$$

Now note that each model can be regarded as the empty (memoryless) model $\{\lambda\}$ to which a number of nodes may have been added. The cost of the empty model is $-\log \alpha_0$, we can also say that the model cost of the first parameter is $-\log \alpha_0$ bits. Our objective is now that, if we add a new node (parameter) to a model, the model cost increases by δ bit, no matter at what level d we add this node. In other words

$$\frac{1 - \alpha_d}{\alpha_d} \cdot \alpha_{d+1}^2 = 2^{-\delta} \tag{31}$$

for $0 \leq d \leq D - 1$, or consequently

$$\left(\frac{1}{\alpha_d}\right) = 2^{-\delta}\left(\frac{1}{\alpha_{d+1}}\right)^2 + 1. \tag{32}$$

If we now assume that $\delta = 0$, which implies that all models that fit into $\mathcal{S} \in \mathcal{C}_D$ have *equal cost*, we find that

$$(\alpha_{D-1})^{-1} = 2, (\alpha_{D-2})^{-1} = 5, (\alpha_{D-3})^{-1} = 26,$$
$$(\alpha_{D-4})^{-1} = 677$$

etc. This yields a cost of $\log 677 = 9.403$ bits for all 677 models in $\mathcal{T}_4$ and 150.448 bits for $D = 8$, etc. Note that the number of models in $\mathcal{C}_D$ grows very fast with D. Incrementing D by one results roughly in squaring the number of models in $\mathcal{C}_D$. The context-tree weighting method is working on all these models simultaneously in a very efficient way!

If we take δ such that $-\log \alpha_0 = \delta$, we obtain model cost $\delta|\mathcal{S}|$, which is *proportional* to the number of parameters $|\mathcal{S}|$. For $D = 1$ we find that

$$\delta = -\log \frac{\sqrt{5} - 1}{2} = 0.694 \text{ bit}$$

for $D = 2$ we get $\delta = 1.047$ bits, $\delta = 1.411$ bits for $D = 4$, and for $D = 8$ we find $\delta = 1.704$ bits, etc.

VIII. Final Remarks

We have seen in Lemma 2 that $P_c(x_1^T|x_{1-D}^0)$ as given by (14) is a weighting over all distributions

$$\prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

corresponding to models $\mathcal{S} \in \mathcal{C}_D$. From (8) we may conclude that

$$\prod_{s \in \mathcal{S}} P_e(a_s, b_s)$$

is a weighting of

$$\prod_{s \in \mathcal{S}} (1 - \theta_s)^{a_s}\theta_s^{b_s}$$

where all components of $\Theta_{\mathcal{S}}$ are assumed to be $(\frac{1}{2}, \frac{1}{2})$-Dirichlet distributed. Therefore, we may say that $P_c(x_1^T|x_{1-D}^0)$ is a weighting over all models $\mathcal{S} \in \mathcal{C}_D$ and all parameter vectors $\Theta_{\mathcal{S}}$, also called a "double mixture" (see [20]). We should stress that the context-tree weighting method induces a certain weighting over all models (see Lemma 2), which can be changed as, e.g., in Section VII in order to achieve specific model redundancy behavior.

The redundancy upper bound in Theorem 2 shows that our method achieves the lower bound obtained by Rissanen (see, e.g. [9, Theorem 1]) for finite-state sources. However, our redundancy bound is in fact stronger, since it holds for all source sequences x_1^T given x_{1-D}^0 and all T, and not only averaged over all source sequences x_1^T given x_{1-D}^0 only for T large enough. Our bound is also stronger in the sense that it is more precise about the terms that tell us about the model redundancy.

The context-tree weighting procedure was presented first at the 1993 IEEE International Symposium on Information Theory in San Antonio, TX (see [22]). At the same time Weinberger, Rissanen, and Feder [21] studied finite-memory tree sources and proposed a method that is based on state estimation. Again an (artificial) constant C and a function $g(t)$ were needed to regulate the selection process. Although we claim that the context-tree method has eliminated all these artificial parameters we must admit that the basic context-tree method, which is described here, has D as a parameter to be specified in advance, making the method work only for models $\mathcal{S} \in \mathcal{C}_D$, i.e., for models with memory not larger than D. It is however possible (see [25]) to modify the algorithm such that there is no constraint on the maximum memory depth D involved. (Moreover, it was demonstrated there that it is not necessary to have access to x_{1-D}^0.) This implementation thus realizes *infinite* context-tree depth D. The storage complexity still remains linear in T. It was furthermore shown in [25] that this implementation of the context-tree weighting method achieves *entropy* for *any* stationary and ergodic source.

In a recent paper, Weinberger, Merhav, and Feder [20] consider the model class containing finite-state sources (and not only bounded memory tree sources). They strengthened the Shtarkov pointwise-minimax lower bound on the individual redundancy ([17, Theorem 1]), i.e., they found a lower bound (equivalent to Rissanen's lower bound for average redundancy [9]) that holds for most sequences in most types. Moreover, they investigated the weighted ("mixing") approach for finite-state sources. Weinberger *et al.* showed that the redundancy for the weighted method achieves their strong lower bound. Furthermore, their paper shows by an example that the state-estimation approach, the authors call this the "plug-in" approach, does not work for all source sequences, i.e., does not achieve the lower bound.

Finite accuracy implementations of the context-tree weighting method in combination with arithmetic coding are studied in [24]. In [23] context weighting methods are described that perform on more general model classes than the one that we have studied here. These model classes are still bounded memory, and the proposed schemes for them are constructive just like the context-tree weighting method that is described here.

Although we have considered only binary sources here, there exist straightforward generalizations of the context-tree weighting method to nonbinary sources (see e.g. [18]).

APPENDIX I
ELIAS ALGORITHM

The first idea behind the Elias algorithm is that to each source sequence x_1^T there corresponds a *subinterval* of $[0,1)$. This principle can be traced back to Shannon [16].

Definition 9: The interval $I(x_1^t)$ corresponding to

$$x_1^t \in \{0,1\}^t, t = 0, 1, \cdots, T$$

is defined as

$$I(x_1^t) \triangleq \left[B(x_1^t), B(x_1^t) + P_c(x_1^t)\right) \tag{33}$$

where

$$B(x_1^t) \triangleq \sum_{\tilde{x}_1^t < x_1^t} P_c(\tilde{x}_1^t)$$

for some ordering over $\{0,1\}^t$.

Note that for $t = 0$ we have that $P_c(\phi) = 1$ and $B(\phi) = 0$ (the only sequence of length 0 is ϕ itself), and consequently, $I(\phi) = [0,1)$. Observe that for any fixed value of $t, t = 0, 1, \cdots, T$, all intervals $I(x_1^t)$ are disjoint, and their union is $[0,1)$. Each interval has a length equal to the corresponding coding probability.

Just like all source sequences, a codeword $c^L = c_1 c_2 \cdots c_L$ can be associated with a subinterval of $[0,1)$.

Definition 10: The interval $J(c^L)$ corresponding to the codeword c^L is defined as

$$J(c^L) \triangleq \left[F(c^L), F(c^L) + 2^{-L}\right) \tag{34}$$

with

$$F(c^L) \triangleq \sum_{l=1,L} c_l 2^{-l}.$$

To understand this, note that c^L can be considered as a binary fraction $F(c^L)$. Since c^L is followed by other codewords, the decoder receives a stream of code digits from which only the first L digits correspond to c^L. The decoder can determine the value that is represented by the binary fraction formed by the total stream $c_1 c_2 \cdots c_L c_{L+1} \cdots$, i.e.

$$F_\infty \triangleq \sum_{l=1,\infty} c_l 2^{-l} \tag{35}$$

where it should be noted that the length of the total stream is not necessarily infinite. Since

$$F(c^L) \le F_\infty < F(c^L) + 2^{-L}$$

we may say that the interval $J(c^L)$ corresponds to the codeword c^L.

To compress a sequence x_1^T, we search for a (short) codeword $c^L(x_1^T)$ whose code interval $J(c^L)$ is contained in the sequence interval $I(x_1^T)$.

Definition 11: The codeword $c^L(x_1^T)$ for source sequence x_1^T consists of

$$L(x_1^T) \triangleq \lceil \log(1/P_c(x_1^T)) \rceil + 1$$

binary digits such that

$$F(c^L(x_1^T)) \triangleq \lceil B(x_1^T) \cdot 2^{L(x_1^T)} \rceil \cdot 2^{-L(x_1^T)} \tag{36}$$

where $\lceil a \rceil$ is the smallest integer $\ge a$. We consider only sequences x_1^T with $P_c(x_1^T) > 0$.

Since

$$F(c^L(x_1^T)) \ge B(x_1^T) \tag{37}$$

and

$$\begin{aligned} F(c^L(x_1^T)) + 2^{-L(x_1^T)} &< B(x_1^T) + 2^{-L(x_1^T)} + 2^{-L(x_1^T)} \\ &\le B(x_1^T) + P_c(x_1^T) \end{aligned} \tag{38}$$

we may conclude that

$$J(c^L(x_1^T)) \subseteq I(x_1^T)$$

and, therefore, $F_\infty \in I(x_1^T)$. Since all intervals $I(x_1^T)$ are disjoint, the decoder can reconstruct the source sequence x_1^T from F_∞. Note that after this reconstruction, the decoder can compute $c^L(x_1^T)$, just like the encoder, and find the location of the first digit of the next codeword. Note also that, since all code intervals are disjoint, no codeword is the prefix of any other codeword. This implies also that the code satisfies the prefix condition. From the definition of the length $L(x_1^T)$ we immediately obtain Theorem 1.

The second idea behind the Elias algorithm, is to order the sequences x_1^t of length t *lexicographically*, for $t, t = 1, \cdots, T$. For two sequences x_1^t and $\tilde{x}_1^t$ we have that $x_1^t < \tilde{x}_1^t$ if and only if there exists a $\tau \in \{1, 2, \cdots, t\}$ such that $x_i = \tilde{x}_i$ for $i = 1, 2, \cdots, \tau - 1$ and $x_\tau < \tilde{x}_\tau$. This lexicographical ordering makes it possible to compute the interval $I(x_1^T)$ sequentially. To do so, we transform the starting interval $I(\phi) = [0,1)$ into $I(x_1), I(x_1 x_2), \cdots$, and $I(x_1 x_2 \cdots x_T)$, respectively. The consequence of the lexicographical ordering over the source sequences is that

$$\begin{aligned} B(x_1^t) &= \sum_{\tilde{x}_1^t < x_1^t} P_c(\tilde{x}_1^t) = \sum_{\tilde{x}_1^{t-1} < x_1^{t-1}} P_c(\tilde{x}_1^{t-1}) \\ &\quad + \sum_{\tilde{x}_t < x_t} P_c(x_1^{t-1}, \tilde{x}_t) \\ &= B(x_1^{t-1}) + \sum_{\tilde{x}_t < x_t} P_c(x_1^{t-1}, \tilde{x}_t). \end{aligned} \tag{39}$$

In other words $B(x_1^t)$ can be computed from $B(x_1^{t-1})$ and $P_c(x_1^{t-1}, X_t = 0)$. Therefore the encoder and the decoder can easily find $I(x_1^t)$ after having determined $I(x_1^{t-1})$, if it is "easy" to determine probabilities $P_c(x_1^{t-1}, X_t = 0)$ and $P_c(x_1^{t-1}, X_t = 1)$ after having processed $x_1 x_2 \cdots x_{t-1}$.

Observe that when the symbol x_t is being processed, the source interval

$$I(x_1^{t-1}) = \left[B(x_1^{t-1}), B(x_1^{t-1}) + P_c(x_1^{t-1})\right)$$

is subdivided into two subintervals

$$I(x_1^{t-1}, X_t=0)=\Big[B(x_1^{t-1}), B(x_1^{t-1})+P_c(x_1^{t-1}, X_t=0)\Big)$$

and

$$I(x_1^{t-1}, X_t=1)=\Big[B(x_1^{t-1})+P_c(x_1^{t-1}, X_t=0), B(x_1^{t-1})+P_c(x_1^{t-1})\Big). \quad (40)$$

The encoder proceeds with one of these subintervals depending on the symbol x_t, therefore $I(x_1^t) \subseteq I(x_1^{t-1})$. This implies that

$$I(x_1^T) \subseteq I(x_1^{T-1}) \subseteq \cdots \subseteq I(\phi). \quad (41)$$

The decoder determines from F_∞ the source symbols $x_1, x_2, \cdots, x_T$, respectively, by comparing F_∞ to thresholds $D(x_1^{t-1})$.

Definition 12: The thresholds $D(x_1^{t-1})$ are defined as

$$D(x_1^{t-1}) \triangleq B(x_1^{t-1}) + P_c(x_1^{t-1}, X_t = 0) \quad (42)$$

for $t = 1, 2, \cdots, T$.

Observe that threshold $D(x_1^{t-1})$ splits up the interval $I(x_1^{t-1})$ in two parts (see (40)). It is the upper boundary point of $I(x_1^{t-1}, X_t = 0)$ but also the lower boundary point of $I(x_1^{t-1}, X_t = 1)$. Since always $F_\infty \in I(x_1^T) \subseteq I(x_1^t)$, we have for $D(x_1^{t-1})$ that

$$\begin{aligned} F_\infty < B(x_1^t) + P_c(x_1^t) &= B(x_1^{t-1}) + P_c(x_1^{t-1}, X_t = 0) \\ &= D(x_1^{t-1}), \quad \text{if } x_t = 0 \end{aligned} \quad (43)$$

and

$$\begin{aligned} F_\infty \geq B(x_1^t) &= B(x_1^{t-1}) + P_c(x_1^{t-1}, X_t = 0\} \\ &= D(x_1^{t-1}), \quad \text{if } x_t = 1. \end{aligned} \quad (44)$$

Therefore, the decoder can easily find x_t by comparing F_∞ to the threshold $D(x_1^{t-1})$; in other words, it can operate sequentially.

Since the code satisfies the prefix condition, it should not be necessary to have access to the complete F_∞ for decoding x_1^T. Indeed, it can be shown that only the first $L(x_1^T)$ digits of the codestream are actually needed.

Appendix II
Properties of the KT-Estimator

Proof: The proof consists of two parts.

1) The fact that $P_e(0, 0) = 1$ follows from

$$\begin{aligned} \int_0^1 \frac{1}{\sqrt{\theta(1-\theta)}}\, d\theta &= \int_0^{\pi/2} \frac{1}{\sin\alpha\cos\alpha}\, d\sin^2\alpha \\ &= \int_0^{\pi/2} 2 d\alpha = \pi. \end{aligned} \quad (45)$$

It is easy to see that

$$P_e(a+1, b) + P_e(a, b+1) = P_e(a, b).$$

We obtain (9) from

$$\begin{aligned} (b+1/2)P_e(a+1, b) &= \frac{b+1/2}{\pi} \int_0^1 (1-\theta)^{a+1/2}\theta^{b-1/2}\, d\theta \\ &= \frac{1}{\pi}\int_0^1 (1-\theta)^{a+1/2} d\theta^{b+1/2} \\ &= -\frac{1}{\pi}\int_0^1 \theta^{b+1/2}\, d(1-\theta)^{a+1/2} \\ &= \frac{a+1/2}{\pi}\int_0^1 (1-\theta)^{a-1/2}\theta^{b+1/2}\, d\theta \\ &= (a+1/2)P_e(a, b+1). \end{aligned} \quad (46)$$

2) Define

$$\Delta(a, b) \triangleq \frac{P_e(a, b)}{\frac{1}{\sqrt{a+b}}\left(\frac{a}{a+b}\right)^a \left(\frac{b}{a+b}\right)^b}. \quad (47)$$

First we assume that $a \geq 1$. Consider

$$\frac{\Delta(a+1, b)}{\Delta(a, b)} = \frac{a^a(a+1/2)}{(a+1)^{a+1}} \cdot \left(\frac{a+b+1}{a+b}\right)^{a+b+1/2}. \quad (48)$$

To analyze (48) we define, for $t \in [1, \infty)$, the functions

$$f(t) \triangleq \ln \frac{t^t(t+1/2)}{(t+1)^{t+1}} \quad \text{and} \quad g(t) \triangleq \ln\left(\frac{t+1}{t}\right)^{t+1/2}. \quad (49)$$

The derivatives of these functions are

$$\frac{df(t)}{dt} = \ln\frac{t}{t+1} + \frac{1}{t+1/2}$$

and

$$\frac{dg(t)}{dt} = \ln\frac{t+1}{t} - \frac{t+1/2}{t(t+1)}. \quad (50)$$

Take

$$\alpha = \frac{1/2}{t+1/2}$$

and observe that $0 < \alpha \leq 1/3$. Then from

$$\begin{aligned} \ln\frac{t}{t+1} = \ln\frac{1-\alpha}{1+\alpha} &= -2\left(\alpha + \frac{\alpha^3}{3} + \frac{\alpha^5}{5} + \cdots\right) \leq -2\alpha \\ &= -\frac{1}{t+1/2} \end{aligned} \quad (51)$$

we obtain that

$$\frac{df(t)}{dt} \leq 0.$$

Therefore

$$\frac{a^a(a+1/2)}{(a+1)^{a+1}} \geq \lim_{a\to\infty} \frac{a^a(a+1/2)}{(a+1)^{a+1}} = \frac{1}{e}. \quad (52)$$

Similarly, from

$$\begin{aligned} \ln\frac{t+1}{t} &= 2\left(\alpha + \frac{\alpha^3}{3} + \frac{\alpha^5}{5} + \cdots\right) \leq 2(\alpha + \alpha^3 + \alpha^5 + \cdots) \\ &= \frac{2\alpha}{1-\alpha^2} = \frac{t+1/2}{t(t+1)} \end{aligned} \quad (53)$$

we may conclude that

$$\frac{dg(t)}{dt} \leq 0.$$

This results in

$$\left(\frac{a+b+1}{a+b}\right)^{a+b+1/2} \geq \lim_{a+b\to\infty}\left(\frac{a+b+1}{a+b}\right)^{a+b+1/2} = e. \tag{54}$$

Combining (52) and (54) yields that

$$\Delta(a+1,b) \geq \Delta(a,b), \quad \text{for } a \geq 1. \tag{55}$$

Next we investigate the case where $a = 0$. Note that this implies that $b \geq 1$, and consider

$$\frac{\Delta(1,b)}{\Delta(0,b)} = \frac{1}{2}\cdot\left(\frac{b+1}{b}\right)^{b+1/2}. \tag{56}$$

If we again use the fact that

$$\frac{dg(t)}{dt} \leq 0$$

we find that

$$\Delta(1,b) \geq \frac{e}{2}\cdot\Delta(0,b). \tag{57}$$

Inequality (55) together with (57), now implies that

$$\Delta(a+1,b) \geq \Delta(a,b), \quad \text{for } a \geq 0. \tag{58}$$

Therefore

$$\Delta(a,b) \geq \Delta(0,1) = \Delta(1,0). \tag{59}$$

The lemma now follows from the observation

$$\Delta(1,0) = \Delta(0,1) = 1/2.$$

It can also easily be proved that $\Delta(a,b) \leq \sqrt{2/\pi}$. Both bounds are tight. ■

APPENDIX III
WEIGHTING PROPERTIES

Proof: We prove by induction that the hypothesis in Lemma 2 holds. For $d = D$ this is true. Next assume that the hypothesis also holds for $0 < d \leq D$. Now consider a node s with $l(s) = d-1$, then

$$\begin{aligned}
P_w^s &= \frac{1}{2}P_e(a_s,b_s) + \frac{1}{2}P_w^{0s}\cdot P_w^{1s} \\
&= \frac{1}{2}P_e(a_s,b_s) \\
&\quad + \frac{1}{2}\left(\sum_{\mathcal{V}\in\mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{V})}\prod_{v\in\mathcal{V}} P_e(a_{v0s},b_{v0s})\right) \\
&\quad \cdot\left(\sum_{\mathcal{W}\in\mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{W})}\prod_{w\in\mathcal{W}} P_e(a_{w1s},b_{w1s})\right) \\
&= 2^{-1}P_e(a_s,b_s) + \sum_{\mathcal{V},\mathcal{W}\in\mathcal{C}_{D-d}} 2^{-1-\Gamma_{D-d}(\mathcal{V})-\Gamma_{D-d}(\mathcal{W})} \\
&\quad \cdot \prod_{u\in\mathcal{V}\times 0\cup\mathcal{W}\times 1} P_e(a_{us},b_{us}) \\
&= \sum_{\mathcal{U}\in\mathcal{C}_{D-d+1}} 2^{-\Gamma_{D-d+1}(\mathcal{U})}\prod_{u\in\mathcal{U}} P_e(a_{us},b_{us}).
\end{aligned} \tag{60}$$

We have used the induction hypothesis in the second step of the derivation. Conclusion is that the hypothesis also holds for $d-1$, and by induction for all $0 \leq d \leq D$. The fact

$$\sum_{\mathcal{U}\in\mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{U})} = 1$$

can be proved similarly if we note that

$$\begin{aligned}
\sum_{\mathcal{U}\in\mathcal{C}_{D-d+1}} 2^{-\Gamma_{D-d+1}(\mathcal{U})} &= \frac{1}{2} + \frac{1}{2}\left(\sum_{\mathcal{V}\in\mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{V})}\right) \\
&\quad \cdot\left(\sum_{\mathcal{W}\in\mathcal{C}_{D-d}} 2^{-\Gamma_{D-d}(\mathcal{W})}\right) \\
&= \frac{1}{2} + \frac{1}{2} = 1.
\end{aligned} \tag{61}$$

■

APPENDIX IV
UPDATING PROPERTIES

Proof: First note that if s is not a suffix of x_{t-D}^{t-1} no descendant of s can be a suffix of x_{t-D}^{t-1}. Therefore, for s and its descendants the a- and b-counts remain the same, and consequently also the estimated probabilities $P_e(a_s,b_s)$, after having observed the symbol x_t. This implies that also the weighted probability P_w^s does not change and (15) holds.

For those $s \in \mathcal{T}_D$ that are a suffix of x_{t-D}^{t-1} we will show that the hypothesis (16) holds by induction. Observe that (16) holds for $l(s) = D$. To see this note that for s such that $l(s) = D$

$$\begin{aligned}
P_w^s(0) + P_w^s(1) &= P_e(a_s+1,b_s) + P_e(a_s,b_s+1) \\
&= P_e(a_s,b_s) = P_w^s(\phi).
\end{aligned} \tag{62}$$

Here we use the following notation :

$$P_w^s(0) = P_w^s(x_1^{t-1}, X_t = 0 | x_{1-D}^0)$$

$$P_w^s(1) = P_w^s(x_1^{t-1}, X_t = 1 | x_{1-D}^0)$$

$$P_w^s(\phi) = P_w^s(x_1^{t-1} | x_{1-D}^0)$$

$$a_s = a_s(x_1^{t-1} | x_{1-D}^0)$$

$$b_s = b_s(x_1^{t-1} | x_{1-D}^0).$$

Next assume that (16) holds for $l(s) = d, 0 < d \leq D$. Now consider nodes corresponding to strings s with $l(s) = d-1$. Then $1s$ is a postfix of x_{t-D}^{t-1} and $0s$ not, or *vice versa*. Let $1s$ be a postfix of x_{t-D}^{t-1}, then

$$\begin{aligned}
P_w^s(0)+P_w^s(1) &= \frac{1}{2}P_e(a_s+1,b_s) + \frac{1}{2}P_w^{0s}(0)\cdot P_w^{1s}(0) \\
&\quad + \frac{1}{2}P_e(a_s,b_s+1) + \frac{1}{2}P_w^{0s}(1)\cdot P_w^{1s}(1) \\
&= \frac{1}{2}P_e(a_s+1,b_s) + \frac{1}{2}P_e(a_s,b_s+1) \\
&\quad + \frac{1}{2}P_w^{0s}(\phi)\cdot P_w^{1s}(0) + \frac{1}{2}P_w^{0s}(\phi)\cdot P_w^{1s}(1)
\end{aligned}$$

$$= \frac{1}{2} P_e(a_s, b_s) + \frac{1}{2} P_w^{0s}(\phi) \cdot \left(P_w^{1s}(0) + P_w^{1s}(1) \right)$$
$$= \frac{1}{2} P_e(a_s, b_s) + \frac{1}{2} P_w^{0s}(\phi) \cdot P_w^{1s}(\phi) = P_w^s(\phi). \tag{63}$$

The induction hypothesis is used to obtain the fourth equality. The second equality follows from (15). The proof is analogous when $0s$ is a postfix of x_{t-D}^{t-1} instead of $1s$. ■

ACKNOWLEDGMENT

This research was carried out in May 1992 while the second author visited the Information Theory group at Eindhoven University. The authors wish to thank the Eindhovens Universiteitsfonds for supporting this visit.

The authors wish to thank P. Volf who participated in research pertaining to Section VII. The comments from the reviewers and the advice of the Associate Editor M. Feder are also acknowledged.

REFERENCES

[1] N. Abramson, *Information Theory and Coding*. New York: McGraw-Hill, 1963, pp. 61–62.
[2] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 73–77, Jan. 1973.
[3] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.
[4] F. Jelinek, *Probabilistic Information Theory*. New York: McGraw-Hill, 1968, pp. 476–489.
[5] R. E. Krichevsky and V. K. Trofimov, "The performance of universal encoding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 199–207, Mar. 1981.
[6] R. Pasco, "Source coding algorithms for fast data compression," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1976.
[7] J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM J. Res. Devel.*, vol. 20, p. 198, 1976.
[8] ——, "A universal data compression system," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 656–664, Sept. 1983.
[9] ——, "Universal coding, information, prediction, and estimation," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 629–636, July 1984.
[10] ——, "Complexity of strings in the class of Markov sources," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 526–532, July 1986.
[11] ——, *Stochastic Complexity in Statistical Inquiry*. Teaneck, NJ : World Scientific, 1989.
[12] J. Rissanen and G. G. Langdon, Jr., "Universal modeling and coding," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 12–23, Jan. 1981.
[13] B. Ya. Ryabko, "Twice-universal coding," *Probl. Inform. Transm.*, vol. 20, no. 3, pp. 24–28, July-Sept. 1984.
[14] ——, "Prediction of random sequences and universal coding," *Probl. Inform. Transm.*, vol. 24, no. 2, pp. 3–14, Apr.–June 1988.
[15] J. P. M. Schalkwijk, "An algorithm for source coding," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 395–399, May 1972.
[16] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948. Reprinted in *Key Papers in the Development of Information Theory*, D. Slepian, Ed. New York: IEEE Press, 1974, pp. 5–18.
[17] Y. M. Shtarkov, "Universal sequential coding of single messages," *Probl. Inform. Transm.*, vol. 23, no. 3, pp. 3–17, July–Sept. 1987.
[18] Tj. J. Tjalkens, Y. M. Shtarkov, and F. M. J. Willems, "Sequential weighting algorithms for multi-alphabet sources," in *6th Joint Swedish-Russian Int. Workshop on Information Theory* (Mölle, Sweden, Aug. 22–27, 1993), pp. 230–239.
[19] M. J. Weinberger, A. Lempel, and J. Ziv, "A sequential algorithm for the universal coding of finite memory sources," *IEEE Trans. Inform. Theory*, vol. 38, pp. 1002–1014, May 1992.
[20] M. J. Weinberger, N. Merhav, and M. Feder, "Optimal sequential probability assignment for individual sequences," *IEEE Trans. Inform. Theory*, vol. 40, pp. 384–396, Mar. 1994.
[21] M. J. Weinberger, J. Rissanen, and M. Feder, "A universal finite memory source," submitted for publication, Aug. 1992; to appear.
[22] F. M. J. Willems, Y. M. Shtarkov, and Tj. J. Tjalkens, "Context tree weighting : A sequential universal source coding procedure for FSMX sources," in *Proc. IEEE Int. Symp. on Information Theory* (San Antonio, TX, Jan. 17–22, 1993), p. 59.
[23] ——, "Context weighting for general finite context sources," submitted to *IEEE Trans. Inform. Theory*, Sept. 1994.
[24] F. M. J. Willems, "The context tree weighting method : Truncated updating," submitted to *IEEE Trans. Inform. Theory*, Aug. 1994.
[25] ——, "Extensions to the context tree weighting method," in *Proc. IEEE Int. Symp. on Information Theory* (Trondheim, Norway, June 27–July 1, 1994), p. 387.