Reflections on "The Context-Tree Weighting Method: Basic Properties"

Frans Willems,* Yuri Shtarkov,† and Tjalling Tjalkens‡

Copyright ©1997 IEEE. Reprinted from IEEE Information Theory Society Newsletter, Vol. 47, No. 1, March 1997.
This material is posted here with permission of the IEEE. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank email message to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

1 Introduction

At the IEEE ISIT in Budapest in 1991 we agreed that Yuri would come to Eindhoven the next year. Supported by the Universiteitsfonds Eindhoven, whose chairman was Jack van Lint at that time, Yuri visited the Information Theory group in May 1992. We decided to investigate universal source coding algorithms for FSMX sources, like e.g. the method proposed by Rissanen in [5]. It was our idea that these algorithms were designed to have a good asymptotical behavior while the non-asymptotical performance, which is important in real life, was not very well understood.

After a while we realized that the idea of selecting relevant contexts was not a good one. This prevented researchers to determine the non-asymptotical behavior of their methods. An obvious alternative was weighting, an almost classical procedure. This way of thinking more or less immediately led to the context-tree weighting (CTW) algorithm.

The CTW algorithm follows from first principles and combines a desirable (asymptotical as well as non-asymptotical) performance with an extremely simple analysis. Therefore, when Michelle Effros asked us to write a "reflections"-paper for the IT Newsletter, we decided to write a mini-course on "universal source coding for tree sources" which at the end focusses on the CTW algorithm. It is the content of the following sections.

2 Sources

Let us consider an information source that generates *binary* sequences of length T. A sequence $x_1^T = x_1 x_2 \cdots x_T$ is generated with (actual) probability $P_a(x_1^T) \triangleq \Pr\{X_1^T = x_1^T\}$.

Example: For a memoryless source $P_a(x_1^T) = \Pi_{t=1,T} P_a(x_n)$ with $P_a(1) = 1 - P_a(0) \triangleq \theta$. The parameter θ, i.e. the probability of generating a 1, has a value in the range $[0, 1]$. If the sequence x_1^T contains a zeroes and b ones then $P_a(x_1^T) = (1-\theta)^a \theta^b$.

*Electrical Engineering Department, Eindhoven University of Technology, Eindhoven, The Netherlands.
†Institute for Problems of Information Transmission, Moscow, Russia.
‡Electrical Engineering Department, Eindhoven University of Technology, Eindhoven, The Netherlands.

3 Codes

A source code assigns to each (possible) source sequence x_1^T a binary codeword $c(x_1^T)$ with length $L(x_1^T)$. We want the lengths of the codewords to be as short as possible, however it is required that the source sequence can be reconstructed from its codeword. Here we only consider prefix codes. In a prefix code no codeword is the prefix of any other codeword. Therefore a prefix code is instantaneously uniquely decodable, i.e. when we receive the last digit of a codeword we know that it is the last one.

Example: Consider for source sequences of length $T = 2$ the assignment in the table below.

x_1^T	$c(x_1^T)$	$L(x_1^T)$
00	0	1
01	10	2
10	110	3
11	111	3

The lengths of the codewords of a binary prefix code satisfy Kraft's inequality (see e.g. Cover and Thomas[1], page 82):

$$\sum_{x_1^T} 2^{-L(x_1^T)} \leq 1. \tag{1}$$

Proof: Let $L^* \triangleq \max_{x_1^T} L(x_1^T)$. A codeword $c(x_1^T)$ has $2^{L^* - L(x_1^T)}$ descendants at level L^*. In a prefix code no two different codewords can have an identical descendant at level L^*. Therefore $\sum_{x_1^T} 2^{L^* - L(x_1^T)} \leq 2^{L^*}$. □

Example: For the code $\{0, 10, 110, 111\}$ we obtain the Kraft sum $2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} \leq 1$.

4 Redundancy

The individual redundancy of a source code, relative to the actual source, for sequence x_1^T is defined as

$$\rho(x_1^T) \triangleq L(x_1^T) - \log \frac{1}{P_a(x_1^T)}. \tag{2}$$

The base of the logarithm is assumed to be 2 here. For the expected redundancy we obtain

$$\begin{aligned} \bar{\rho} &\triangleq \sum_{x_1^T} P_a(x_1^T) L(x_1^T) - \sum_{x_1^T} P_a(x_1^T) \log \frac{1}{P_a(x_1^T)} \\ &= \bar{L} - H(X_1^T), \end{aligned} \tag{3}$$

i.e. the difference between the expected codeword length $\bar{L}$ and the entropy $H(X_1^T)$ of the source sequences.

The smallest possible expected redundancy is 0.

Proof: Rewrite

$$\bar{\rho} = \sum_{x_1^T} P_a(x_1^T) \log \frac{P_a(x_1^T)}{2^{-L(x_1^T)}/K} + \log \frac{1}{K}, \qquad (4)$$

where $K = \sum_{x_1^T} 2^{-L(x_1^T)}$. The first term on the right hand side is a divergence and can not be negative (see [1] page 26). The second term is non-negative by Kraft's inequality (1). □

Note that we achieve $\bar{\rho} = 0$ only if the codeword length $L(x_1^T) = \log 1/P_a(x_1^T)$ for all x_1^T. Therefore we call such codeword lengths *ideal*. It is our intention to design codes that approach these ideal codeword lengths as closely as possible.

Example: Let $T = 2$. Consider the code $\{0, 10, 110, 111\}$ and a memoryless source with $\theta = 0.2$. The table below lists the individual redundancies for this code.

x_1^T	$c(x_1^T)$	$L(x_1^T)$	$P_a(x_1^T)$	$\rho(x_1^T)$
00	0	1	0.64	0.356
01	10	2	0.16	-0.644
10	110	3	0.16	0.356
11	111	3	0.04	-1.644

The expected redundancy $\bar{\rho} = 0.116$.

5 Arithmetic coding

Assume a lexicographical ordering over the sequences. Now let $Q_a(x_1^T) \triangleq \sum_{\tilde{x}^T < x_1^T} P_a(\tilde{x}^T)$ be the cumulative probability of x_1^T. Then associate to a source sequence x_1^T the source interval $I(x_1^T) = [Q_a(x_1^T), Q_a(x_1^T) + P_a(x_1^T))$. Note that all source intervals $I(x_1^T)$ are disjoint, and that their union is $[0, 1)$.

Also to a codeword there corresponds an interval. To codeword c having length L there corresponds an interval $J(c) = [.c, .c + 2^{-L})$, where $.c$ is the number obtained by considering c as a binary fraction. Note that a codeword can only be the prefix of an other codeword if its code interval contains the code interval of the other codeword.

The idea of arithmetic coding is to choose for a given source sequence x_1^T the codeword $c(x_1^T)$ with a code interval $J(c(x_1^T))$ inside $I(x_1^T)$. This is obtained by taking

$$\begin{aligned} L(x_1^T) &= \lceil \log 1/P_a(x_1^T) \rceil + 1, \\ \text{and } .c &= \lceil Q_a(x_1^T) \cdot 2^{L(x_1^T)} \rceil 2^{-L(x_1^T)}, \end{aligned} \qquad (5)$$

where $\lceil z \rceil$ is the smallest integer $\geq z$. This follows from

$$\begin{aligned} &\lceil Q_a(x_1^T) \cdot 2^{L(x_1^T)} \rceil 2^{-L(x_1^T)} + 2^{-L(x_1^T)} \\ &< Q + 2^{1-L} \\ &= Q + 2^{-\lceil \log 1/P_a \rceil} \leq Q + P_a, \end{aligned} \qquad (6)$$

in which we simplified the notation a little bit.

Example: Let $T = 2$ and consider again a memoryless source with $\theta = 0.2$. The source intervals, codewords, and redundancies are put in the table.

x_1^T	$P_a(x_1^T)$	$Q_a(x_1^T)$	$L(x_1^T)$	$c(x_1^T)$	$\rho(x_1^T)$
00	0.64	0.00	2	00	1.356
01	0.16	0.64	4	1011	1.356
10	0.16	0.80	4	1101	1.356
11	0.04	0.96	6	111110	1.356

Since $L(x_1^T) < \log 1/P_a(x_1^T) + 2$ we may conclude that arithmetic coding achieves codeword lengths within 2 bit from the ideal codeword length, or $\rho(x_1^T) < 2$ for all x_1^T. This implies also that $\bar{\rho} < 2$ or $\bar{L} < H(X_1^T) + 2$.

For the cumulative probability we can write (Elias):

$$Q_a(x_1^T) = \sum_{t=1, T|x_t=1} P_a(x_1 x_2 \cdots x_{t-1} 0). \qquad (7)$$

This makes it easy to compute this probability (and $P_a(x_1^T)$) if, after knowing $x_1 x_2 \cdots x_{t-1}$ and $P_a(x_1 x_2 \cdots x_{t-1})$ we can easily compute $P_a(x_1 x_2 \cdots x_{t-1} 0)$ and $P_a(x_1 x_2 \cdots x_{t-1} 1)$.

Example: For $T = 3$ and a memoryless source having $\theta = 0.2$ we get e.g. $Q_a(101) = P_a(0) + P_a(100) = 0.8 + 0.2 \cdot 0.8 \cdot 0.8 = 0.928$ and $P_a(101) = 0.2 \cdot 0.8 \cdot 0.2 = 0.032$.

If we use, instead of the actual distribution $P_a(x_1^T)$, an arbitrary coding distribution $P_c(x_1^T)$, we obtain codeword lengths for which

$$L(x_1^T) < \log \frac{1}{P_c(x_1^T)} + 2. \qquad (8)$$

We say that the *coding redundancy* is smaller than 2 bits. Note that it is necessary that $P_c(x_1^T) > 0$ for x_1^T such that $P_a(x_1^T) > 0$.

Note that arithmetic coding achieves codeword lengths that are very close to what we want (i.e. $\log 1/P_c$) and that no tables are needed to store the code (the codeword is computed from the source sequence and vice versa). Issues regarding implementation of this technique are considered e.g. by Rissanen[4] and Pasco[3]. We now have to concentrate on constructing good coding distributions.

6 One unknown parameter

Suppose that our actual source is memoryless with a, to us unknown, parameter $\theta \in [0, 1]$. Is it possible now to design a code which has acceptable individual redundancies for all sequences x_1^T relative to all possible sources?

The answer to this question turns out to be affirmative. Just apply arithmetic coding with a coding distribution equal to an estimated distribution $P_e(x_1^T)$ formulated by Krichevsky and Trofimov[2]. This KT-distribution is defined by the following conditional probability:

$$P_e(X_t = 1 | x_1^{t-1}) = \frac{b + 1/2}{a + b + 1}, \qquad (9)$$

for x_1^{t-1} containing a zeroes and b ones. Our estimated block probability $P_e(x_1^T)$ is the product of conditional probabilities as in (9).

Example: The estimated probability $P_e(01110) = \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{3}{6} \cdot \frac{5}{8} \cdot \frac{3}{10} = \frac{3}{256}$. Similarly $Q_e(01110) = \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{3}{6} + \frac{1}{2} \cdot \frac{1}{4} \cdot \frac{3}{6} \cdot \frac{3}{8} = \frac{59}{128}$ (see figure 1).

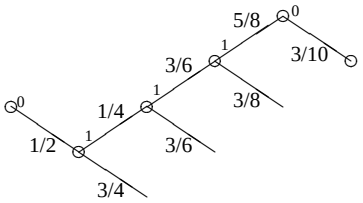

Figure 1: Computation of $P_e(01110)$ and $Q_e(01110)$.

The estimated block probability of a sequence containing a zeroes and b ones is

$$P_e(a,b) = \frac{1/2 \cdot \ldots \cdot (a-1/2) \cdot 1/2 \cdot \ldots \cdot (b-1/2)}{1 \cdot 2 \cdot \ldots \cdot (a+b)}. \tag{10}$$

Example: Tabulated below is $P_e(a,b)$ for several (a,b).

a \ b	1	2	3	4	5
0	1/2	3/8	5/16	35/128	63/256
1	1/8	1/16	5/128	7/256	21/1024
2	1/16	3/128	3/256	7/1024	9/2048
3	5/128	3/256	5/1024	5/2048	45/32768

What is the redundancy of this KT-estimated distribution? Consider a sequence x_1^T with a zeroes and b ones, then from (8) we may conclude that

$$L(x_1^T) < \log 1/P_e(a,b) + 2. \tag{11}$$

Therefore

$$\begin{aligned} \rho(x_1^T) &< \log \frac{1}{P_e(a,b)} + 2 - \log \frac{1}{(1-\theta)^a \theta^b} \\ &= \log \frac{(1-\theta)^a \theta^b}{P_e(a,b)} + 2 \\ &\leq \frac{1}{2}\log(a+b) + 3, \end{aligned} \tag{12}$$

where we used lemma 1 of [7] to upper bound the $\log P_a/P_e$-term. This term, called the *parameter redundancy*, is never larger than $\frac{1}{2}\log(a+b)+1$. Hence the individual redundancy is not larger than $\frac{1}{2}\log T + 3$ for all x_1^T and all $\theta \in [0,1]$.

Example: Let $T = 1024$, then the codeword length is not larger than the ideal codeword length plus $\frac{1}{2}\log 1024 + 3 = 8$ bit.

7 Tree sources

If a source is memoryless, each new source symbol is generated according to the same parameter θ. In a more complex situation we can assume that the parameter for generating the next symbol depends on the most recent source symbols. A tree source is a nice concept to describe such sources. A tree source consists of a set $\mathcal{S}$ of suffixes that together form a tree (see figure 2). To each suffix (leaf) s in the tree there corresponds a parameter θ_s. The probability of the next source symbol being one depends on the suffix in $\mathcal{S}$ of the semi-infinite sequence of past source symbols.

We clearly want to distinguish between parameters and model. The model is the mechanism that enables the parameters, i.e. the suffix set (or tree).

A context of a source symbol x_t is a suffix of the semi-infinite sequence $\cdots x_{t-2}x_{t-1}$ that precedes it.

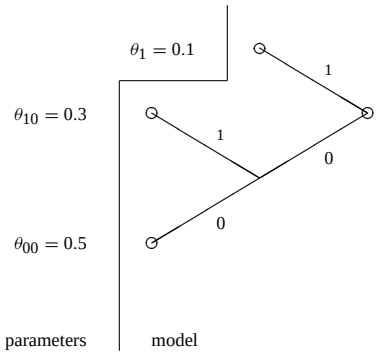

Figure 2: Tree source with parameters and model.

Example: Let $\mathcal{S} \triangleq \{00, 10, 1\}$ and $\theta_{00} = 0.5, \theta_{10} = 0.3$, and $\theta_1 = 0.1$ then

$$\begin{aligned} P_a(X_t = 1|\cdots, X_{t-2} = 0, X_{t-1} = 0) &= 0.5, \\ P_a(X_t = 1|\cdots, X_{t-2} = 1, X_{t-1} = 0) &= 0.3, \\ P_a(X_t = 1|\cdots, X_{t-1} = 1) &= 0.1. \end{aligned} \tag{13}$$

For each source symbol x_t we start in the root of the tree (see figure 2) and follow a path determined by past symbols $x_{t-1}, x_{t-2}, \cdots$. We always end up in a leaf. There we find the parameter for generating x_t.

8 Known model, unknown parameters

In this section we assume that we know the tree model of the actual source but not its parameters. Can we find a good coding distribution for this case?

In principle this problem is quite simple. Since we know the model we can for each source symbol determine its suffix. All symbols that correspond to the same suffix $s \in \mathcal{S}$ form a memoryless subsequence whose statistic is determined by an unknown parameter θ_s. For this subsequence we simply use the KT-estimator. The estimated probability of the entire source sequence is the product of the KT-estimates for the subsequences and hence by (8), we obtain

$$L(x_1^T) < \log 1/\prod_{s\in\mathcal{S}} P_e(a_s, b_s) + 2. \tag{14}$$

Example: Let $\mathcal{S} = \{00, 10, 1\}$. The estimated probability

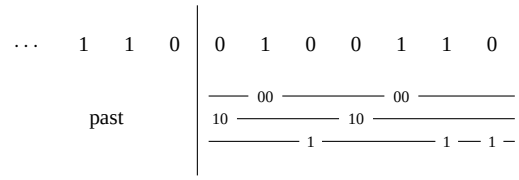

Figure 3: A sequence, its subsequences, and the past.

of the sequence 0100110 given the past $\cdots 110$ (see figure 3) is $P_e^{\mathcal{S}}(0100110|\cdots 110) = \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{3}{4}\cdot\frac{1}{4}\cdot\frac{3}{6} = \frac{3}{8}\cdot\frac{3}{8}\cdot\frac{1}{16} = \frac{9}{1024}$, where $\frac{3}{8}$, $\frac{3}{8}$, and $\frac{1}{16}$ are the probabilities of the subsequences 11, 00, and 010, corresponding to the leaves 00, 10, and 1 respectively.

Again, what is the redundancy of this estimated distribution? Consider a sequence x_1^T with subsequence corresponding to leaf s having a_s zeroes and b_s ones. Then, using

(14) we obtain

$$\begin{aligned}\rho(x_1^T) &< \sum_{s\in\mathcal{S}} \log \frac{(1-\theta_s)^{a_s}\theta_s^{b_s}}{P_e(a_s,b_s)} + 2, \\ &\leq \sum_{s\in\mathcal{S}|a_s+b_s>0} \left(\frac{1}{2}\log(a_s+b_s)+1\right) + 2 \\ &\leq |\mathcal{S}|\gamma(\frac{T}{|\mathcal{S}|}) + 2, \end{aligned} \tag{15}$$

where we again used lemma 1 in [7] for bounding the parameter redundancies for all the subsequences. Convexity is used to obtain a bound on the sum of the logarithms in terms of the γ function which is defined as

$$\gamma(z) \triangleq \begin{cases} z & \text{for } 0 \leq z < 1 \\ \frac{1}{2}\log z + 1 & \text{for } z \geq 1. \end{cases} \tag{16}$$

Note that (15) holds for all x_1^T and all $\theta_s \in [0,1]$ for $s \in \mathcal{S}$.

Example: Let $\mathcal{S} = \{00, 10, 1\}$ and $T = 1024$, then the codeword is never longer than the ideal codeword length plus $\frac{3}{2}\log\frac{1024}{3} + 3 + 2 = 17.623$ bit.

9 Weighting alternatives

Suppose that $P_c^1(x_1^T)$ is a good coding distribution for source 1 and $P_c^2(x_1^T)$ for source 2. Then the *weighted* distribution

$$P_c^w(x_1^T) \triangleq \frac{P_c^1(x_1^T) + P_c^2(x_1^T)}{2} \tag{17}$$

is a good coding distribution for both source 1 and 2.

Proof: Let $i \in \{1,2\}$, then

$$L(x_1^T) < \log\frac{1}{P_c^w(x_1^T)} + 2 \tag{18}$$

$$\leq \log\frac{2}{P_c^i(x_1^T)} + 2 = \log\frac{1}{P_c^i(x_1^T)} + 3. \tag{19}$$

□

So the bound on the codeword length increases (see (8)) by 1 bit. In practice the increase is far less, especially if $P_c^1(x_1^T)$ and $P_c^2(x_1^T)$ are approximately equal.

Note that, if after observing x_1^T we *select* the i that minimizes $P_c^i(x_1^T)$, we loose exactly 1 bit. This bit is now needed to specify the source index.

Example: Suppose sources 1 and 2 are memoryless with parameters $\theta_1 = 0.8$ and $\theta_2 = 0.4$. Then $P_w(X_1 = 1) = 1/2(0.8 + 0.4) = 0.6$, $P_w(X_1 = 1, X_2 = 1) = 1/2(0.8 \cdot 0.8 + 0.4 \cdot 0.4) = 0.40$, and $P_w(X_1 = 1, X_2 = 1, X_3 = 1) = 1/2(0.8 \cdot 0.8 \cdot 0.8 + 0.4 \cdot 0.4 \cdot 0.4) = 0.288$. Hence $P_w(X_3 = 1 | X_1 = 1, X_2 = 1) = 0.288/0.4 = 0.72$ which is close to θ_1. Similarly, $P_w(X_3 = 1 | X_1 = 0, X_2 = 0) = 0.44$ is close to θ_2.

10 Unknown model

10.1 Context tree

If the actual model of our tree source is not known, we can use a *context tree* to compute an appropriate coding distribution. A context tree (see figure 4) consists of nodes that correspond to contexts s up to a certain depth D. The root λ of the context tree corresponds to the empty context. Each node s in the context tree is associated with the subsequence of source symbols that occurred after context s.

Example: Suppose that the source generated the sequence

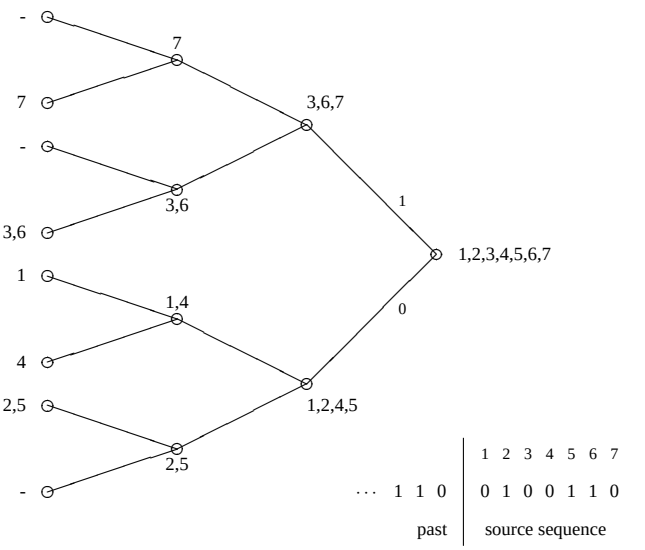

Figure 4: The context tree splits up the source sequence.

0100110 while the past symbols were $\cdots 110$. Then the source symbols are partitioned by the context tree, see figure 4.

10.2 CTW algorithm

We are now ready to formulate the context-tree weighting algorithm.

We start by making the practical assumption that in node s of the context tree we only store a_s and b_s, i.e. the number of zeroes and ones in the subsequence associated with context s.

First we consider a leaf s of the context tree, i.e. a node (context) at depth D. Since only a_s and b_s are available in this node, we can assume nothing more than that the subsequence associated with node s is memoryless and that $P_e(a_s, b_s)$ is a good weighted probability for it, i.e.

$$P_w^s \triangleq P_e(a_s, b_s) \text{ if depth}(s) = D. \tag{20}$$

These weighted probabilities are needed to start the recursion described in the next paragraph.

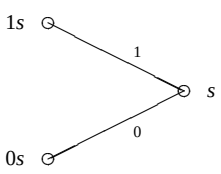

Figure 5: Parent node s and its children $0s$ and $1s$.

For an internal node s in the context tree the following argument holds. Suppose that we have good weighted probabilities P_w^{0s} and P_w^{1s} for the subsequences associated with nodes $0s$ and $1s$, the children (see figure 5) of s. Then for the subsequence associated with context s there are two possible alternatives. It could be memoryless, in which case

$P_e(a_s, b_s)$ would be good coding probability for it. Or, it could not be memoryless, and then splitting up the sequence in the two subsequences that are associated with 0s and 1s would be necessary, and the product of the weighted probabilities P_w^{0s} and P_w^{1s} could serve as a good coding probability. Following the philosophy in section 9 it is completely clear that we should just weight these two alternatives:

$$P_w^s \triangleq \frac{P_e(a_s, b_s) + P_w^{0s} \cdot P_w^{1s}}{2} \text{ if depth}(s) < D. \qquad (21)$$

In the root λ of the context tree we will now find weighted probabilities which can be used as coding probabilities for arithmetic encoding and decoding.

Example: Again suppose that the source generated the sequence 0100110 while the past symbols were $\cdots 110$. This results in the counts a_s and b_s, and weighted probabilities P_w^s, for s with depth $\leq D$, which are depicted in the context tree in figure 6.

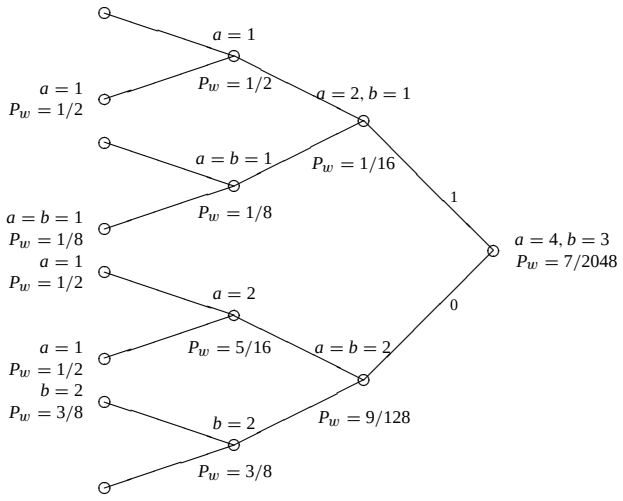

Figure 6: Weighted context tree for source sequence 0100110 with past $\cdots 110$.

10.3 Analysis

We start this subsection by taking a look at an example.

Example: Suppose that $\mathcal{S} = \{00, 10, 1\}$ is the model of the actual source. The depth D of our context tree is 3. Then for the leaves of this model we can lower bound the weighted probabilities P_w^s by the $P_e(a_s, b_s)$-terms, i.e.

$$\begin{aligned} P_w^{00} &\geq (1/2) P_e(a_{00}, b_{00}), \\ P_w^{10} &\geq (1/2) P_e(a_{10}, b_{10}), \\ P_w^{1} &\geq (1/2) P_e(a_{1}, b_{1}), \end{aligned} \qquad (22)$$

while for the internal nodes we use the $P_w^{01} \cdot P_w^{1s}$-term as lower bound, hence

$$\begin{aligned} P_w^{0} &\geq (1/2) P_w^{00} \cdot P_w^{10} \\ &\geq (1/8) P_e(a_{00}, b_{00}) P_e(a_{10}, b_{10}) \text{ and} \\ P_w^{\lambda} &\geq (1/2) P_w^{0} \cdot P_w^{1} \\ &\geq (1/32) P_e(a_{00}, b_{00}) P_e(a_{10}, b_{10}) P_e(a_{1}, b_{1}). \end{aligned} \qquad (23)$$

From the example we may conclude that we loose a factor $1/2$ in all $|\mathcal{S}|$ leaves and $|\mathcal{S}| - 1$ internal nodes of the actual model, hence

$$P_w^{\lambda} \geq 2^{1-2|\mathcal{S}|} \cdot \prod_{s \in \mathcal{S}} P_e(a_s, b_s). \qquad (24)$$

Using (8) we find that

$$L(x_1^T) < 2|\mathcal{S}| - 1 + \log 1/ \prod_{s \in \mathcal{S}} P_e(a_s, b_s) + 2, \qquad (25)$$

which is $2|\mathcal{S}| - 1$ bits more than the bound in (14), where the model was known. Therefore also the bound on the individual redundancy is $2|\mathcal{S}| - 1$ bits larger than the bound in (15), i.e.:

$$\rho(x_1^T) < 2|\mathcal{S}| - 1 + |\mathcal{S}| \gamma(\frac{T}{|\mathcal{S}|}) + 2. \qquad (26)$$

The increase of $2|\mathcal{S}| - 1$ bits can be considered as the cost of not knowing the model, i.e. the *model redundancy*. Note that (26) holds for all x_1^T and all $\theta_s \in [0, 1]$ for $s \in \mathcal{S}$ for all models $\mathcal{S}$ that fit in our context tree.

10.4 Optimality

The expected redundancy behavior of the CTW method achieves the asymptotic lower bound determined by Rissanen in [6]. This lower bound states that roughly $(1/2) \log T$ bits per parameter is the minimum possible expected redundancy for $T \to \infty$.

10.5 MDL behavior

So far we have only compared the CTW codeword length resulting from our source sequence x_1^T to the ideal codeword length relative to the actual source. We have shown that this codeword length is upper bounded by the ideal codeword length plus an upper bound for the individual redundancy in terms of the sequence length T and the number of parameters $|\mathcal{S}|$ of the actual source.

If however this x_1^T was generated by some other tree source the same bound on the codeword length applies but now with the ideal codelength and redundancy as determined by the other source. Hence upper bounds like this, for all tree sources, hold for the CTW codeword length and the CTW algorithm minimizes, over all tree sources, the sum of the corresponding ideal codeword length and redundancy. This can be considered as minimum description length (MDL) behavior, if we realize that the redundancy is needed to describe the source model and parameters.

10.6 Complexity

In section 5 we have seen that we can use arithmetic coding if, after $x_1 x_2 \cdots x_{t-1}$ is processed and $P_c(x_1 x_2 \cdots x_{t-1})$ is known, it is easy to compute $P_c(x_1 x_2 \cdots x_{t-1} 0)$ and $P_c(x_1 x_2 \cdots x_{t-1} 1)$. Does this hold for the CTW algorithm? Fortunately it does! See the example below.

Example: Suppose that the source has already generated the sequence 0100110 while the past symbols were $\cdots 110$. This resulted in the weighted context tree in figure 6. In the root of the context tree we found the coding probability $P_c(0100110|\cdots 110) = P_w^{\lambda} = 7/2048$. For processing the next source symbol we must

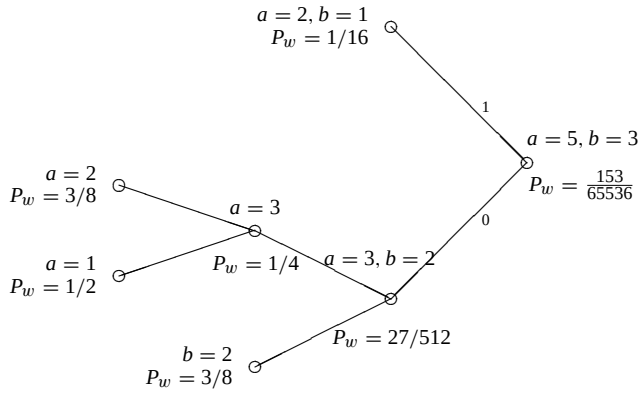

Figure 7: Updated path of the weighted context tree for 0100110 followed by a 0 with past $\cdots 110$.

compute the coding probability $P_c(01001100|\cdots 110)$. This done by (i) incrementing a_s, (ii) updating $P_e(a_s, b_s)$, and (iii) updating P_w^s, for all contexts $s \in \{110, 10, 0, \lambda\}$, i.e. along the path in the context tree determined by the symbols preceding the next source symbol. The results of these computations are shown in figure 7. Doing so we find $P_c(01001100|\cdots 110) = P_w^\lambda$ in the root of the context tree. In a similar way $P_c(01001101|\cdots 110)$ can be determined.

We conclude by stating that the number of operations necessary for processing all T source symbols is linear in T. Moreover, since we only need to store nodes in the context tree that actually did occur, and since for each symbol we can visit not more than $D+1$ nodes, the amount of memory needed to compress x_1^T grows not faster than linear in T.

11 Text compression

Application of the binary CTW method for compression of ASCII sequences is possible after *decomposing* the ASCII symbols into 7 binary digits. Moreover we use 7 context trees. Tree$_j$, where $j = 1, 7$, is used for coding all binary digits occurring at position j in an ASCII. The (longest) context of digit j are the digits $j-1, \cdots, 1$ of the current ASCII preceded by the 7 digits in the most recent ASCII, preceded by the 7 digits in the second most recent ASCII, up to the M'th most recent ASCII. Digit 1 in an ASCII is the most significant one, and is encoded first, etc. By decomposing we allow different tree models for all 7 digits. This may reduce the total number of parameters and thus the redundancy.

The next problem that has to be solved is that the parameter redundancy depends on the number of parameters, however, after decomposition many of these parameters are 0 or 1, possibly because of non-occurring ASCII symbols. Therefore we use, instead of the KT-estimator, the *"zero-redundancy" estimator*, which is defined as

$$P_e^z(a,b) \stackrel{\Delta}{=} \frac{1}{2}P_e(a,b) + \frac{1}{4}\vartheta(a=0) + \frac{1}{4}\vartheta(b=0), \quad (27)$$

where $P_e(a,b)$ is the KT-estimator and $\vartheta(\text{true}) \stackrel{\Delta}{=} 1$ and $\vartheta(\text{false}) \stackrel{\Delta}{=} 0$. This leads to a redundancy of not more than 2 bits for deterministic subsequences, i.e. sequences with $a = 0$ or $b = 0$, and $(1/2)\log\tau + 2$ for non-deterministic ones having length τ, where $\tau = a + b$.

Our final problem deals with model redundancy. Note that context tree$_j$, for some $j = 1, 7$, "fits" a model to the data (digits occurring at position j) which is a binary tree. Such a tree can have leaves also at positions inside an ASCII symbol. This seems not very useful. By allowing leaves only at ASCII borders we decrease the model redundancy. This is accomplished by *weighting only at ASCII borders*, i.e. taking $P_w^s = P_w^{0s} \cdot P_w^{1s}$ in nodes inside an ASCII and $P_w^s = (P_e^z(a_s, b_s) + P_w^{0s} \cdot P_w^{1s})/2$ in nodes on ASCII borders.

ASCII CTW for $M = 12$ achieves 1.825 bit/sym on the file `bib`, 2.180 bit/sym on `book1`, 1.875 bit/sym on `book2`, 2.346 bit/sym on `news`, 2.290 bit/sym on `paper1`, 2.232 bit/sym on `paper2`, and 2.336 bit/sym on `progc` of the Calgary corpus.

12 Conclusion

We believe that context-tree weighting simplified the theory and practice of statistical data compression methods. It is important to distinguish between model and parameters and to realize that to both of them there corresponds a redundancy term. Good algorithms take care of both redundancies. The model redundancy of CTW is optimal in the rather weak sense that we can decrease the redundancy for some models only by increasing the redundancy of other models. This is a consequence of weighting. There are other weightings that result in other model redundancy profiles, however CTW has the nice property that the model redundancy is (almost) proportional to the number of parameters.

The CTW method is generally considered to be rather complex. A state-of-the-art implementation requires 32 MByte of RAM. Today this may seem a lot, however for sure, in ten years from now this is "peanuts." A challenging problem is to find methods that improve the compression rate of e.g. CTW by making use of the huge amounts of memory that will be available in the future. Of particular interest are of course methods that allow parallel implementation. We hope that the mini-course presented here will be a starting point for people interested in achieving this goal.

Acknowledgements

Meir Feder, assoc. editor for source coding, nominated [7] for the IT Best Paper Award. KPN Research financed CTW implementation research. Paul Volf joined us in our text compression efforts. Harry Creemers provided us with computing facilities. Yuri's visits were supported by the Universiteitsfonds Eindhoven and INTAS 94-469. Thanks!

References

[1] T.M. Cover and J.A. Thomas, *Elements of Information Theory.* New York : John Wiley, 1991.

[2] R.E. Krichevsky and V.K. Trofimov, "The Performance of Universal Encoding," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 199-207, March 1981.

[3] R. Pasco, *Source Coding Algorithms for Fast Data Compression,* Ph.D. thesis, Stanford University, 1976.

[4] J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM J. Res. Devel.*, vol. 20, p. 198-203, 1976.

[5] J. Rissanen, “A Universal Data Compression System,” *IEEE Inform. Theory,* vol. IT-29, pp. 656-664, September 1983.

[6] J. Rissanen, “Universal Coding, Information, Prediction, and Estimation,” *IEEE Inform. Theory,* vol. IT-30, pp. 629-636, July 1984.

[7] F.M.J. Willems, Y.M. Shtarkov and Tj.J. Tjalkens, “The Context-Tree Weighting Method: Basic Properties,” *IEEE Trans. Inform. Theory,* vol. IT-41, pp. 653-664, May 1995.