

Superior Guarantees for Sequential Prediction and Lossless Compression via Alphabet Decomposition

Ron Begleiter and Ran El-Yaniv

ronbeg@cs.technion.ac.il

rani@cs.technion.ac.il

Department of Computer Science

Technion - Israel Institute of Technology

Haifa 32000, Israel

Abstract

We present worst case bounds for the learning rate of a known prediction method that is based on hierarchical applications of binary Context Tree Weighting (CTW) predictors. A heuristic application of this approach that relies on Huffman’s alphabet decomposition is known to achieve state-of-the-art performance in prediction and lossless compression benchmarks. We show that our new bound for this heuristic is tighter than the best known performance guarantees for prediction and lossless compression algorithms in various settings. This result substantiates the efficiency of this hierarchical method and provides a compelling explanation for its practical success. In addition, we present the results of a few experiments that examine other possibilities for improving the multi-alphabet prediction performance of CTW-based algorithms.

1. Introduction

Sequence prediction and entropy estimation are fundamental tasks in numerous machine learning and data mining applications. Here we consider a standard discrete sequence prediction setting where performance is measured via the log-loss (self-information). It is well known that this setting is intimately related to lossless compression, where in fact high quality prediction is essentially equivalent to high quality lossless compression.

Despite the major interest in sequence prediction and the existence of a number of *universal* prediction algorithms, some fundamental issues related to learning from finite (and small) samples are still open. One issue that motivated the current research is that the finite-sample behavior of prediction algorithms is still not sufficiently understood.

Among the numerous compression and prediction algorithms there are very few that offer both finite sample guarantees and good practical performance. The *context tree weighting* (CTW) method of Willems et al. (1995) is a member of this exclusive family of algorithms. The CTW algorithm is an “ensemble method,” mixing the predictions of many underlying variable order Markov models (VMMs), where each such model is constructed using zero-order conditional probability estimators. The algorithm is *universal* with respect to the class of bounded-order VMM tree-sources. Moreover, the algorithm has a finite sample point-wise redundancy bound (for any particular sequence).

The high practical performance of the original CTW algorithm is most apparent when applied to *binary* prediction problems, in which case it uses the well-known (binary) KT-estimator (Krichevsky and Trofimov, 1981). When the algorithm is applied to non-binary

prediction/compression problems (using the multi-alphabet KT-estimator), its empirical performance is mediocre compared to the best known results (Tjalkens et al., 1997). Nevertheless, a clever *alphabet decomposition* heuristic, suggested by Tjalkens et al. (1994) and further developed by Volf (2002), does achieve state-of-the-art compression and prediction performance on standard benchmarks (see, e.g., Volf, 2002; Sadakane et al., 2000; Shkarin, 2002; Begleiter et al., 2004). In this approach the multi-alphabet problem is hierarchically decomposed into a number of binary prediction problems. We term the resulting procedure “the DECO algorithm.” Volf suggested applying the DECO algorithm using Huffman’s tree as the decomposition structure, where the tree construction is based on letter frequencies. We are not aware of any previous compelling explanation for the striking empirical success of DECO.

Our main contribution is a general worst case redundancy bound for algorithm DECO applied with any alphabet decomposition structure. The bound proves that the algorithm is *universal* with respect to VMMS. A specialization of the bound to the case of Huffman decompositions results in a tight redundancy bound. To the best of our knowledge, this new bound is the sharpest available for prediction and lossless compression for sufficiently large alphabets and sequences.

We also present a few empirical results that provide some insight into the following questions: (1) Can we improve on the Huffman decomposition structure using an optimized decomposition tree? (2) Can other, perhaps “flat” types of alphabet decomposition schemes outperform the hierarchical approach? (3) Can standard CTW multi-alphabet prediction be improved with other types of (non-KT) zero-order estimators?

Before we start with the technical exposition, we introduce some standard terms and definitions. Throughout the paper, Σ denotes a finite alphabet with $k = |\Sigma|$ symbols. Suppose we are given a sequence $\mathbf{x}_1^n = x_1, x_2, \dots, x_n$. Our goal is to generate a probabilistic prediction $\hat{P}(x_{n+1}|\mathbf{x}_1^n)$ for the next symbol given the previous symbols. Clearly this is equivalent to being able to estimate the probability $\hat{P}(\mathbf{x}_1^n)$ of any complete sequence, since $\hat{P}(x_{n+1}|\mathbf{x}_1^n) = \hat{P}(x_1^{n+1})/\hat{P}(\mathbf{x}_1^n)$ (provided that the marginality condition $\sum_{\sigma} \hat{P}(\mathbf{x}_1^n \sigma) = \hat{P}(\mathbf{x}_1^n)$ holds).

We consider a setting where the performance of the prediction algorithm is measured with respect to the best predictor in some reference, which we call here a *comparison class*. In our case the comparison class is the set of all variable order Markov models (see details below). Let ALG be a prediction algorithm that assigns a probability estimate $P_{\text{ALG}}(\mathbf{x}_1^n)$ for any given \mathbf{x}_1^n . The *point-wise redundancy* of ALG with respect to the predictor P and the sequence \mathbf{x}_1^n is $R_{\text{ALG}}(\mathbf{x}_1^n, P) = \log P(\mathbf{x}_1^n) - \log P_{\text{ALG}}(\mathbf{x}_1^n)$. The per-symbol point-wise redundancy is $\frac{1}{n}R_{\text{ALG}}(\mathbf{x}_1^n, P)$. ALG is called *universal* with respect to a comparison class \mathcal{C} , if

$$\lim_{n \rightarrow \infty} \sup_{P \in \mathcal{C}} \max_{\mathbf{x}_1^n} \frac{1}{n} R_{\text{ALG}}(\mathbf{x}_1^n, P) = 0. \quad (1)$$

2. Preliminaries

This section presents the relevant technical background for the present work. The contextual background appears in Section 7. We start by presenting the class of variable order Markov suffix tree-sources. We then describe the CTW algorithm and discuss some of its known

properties and performance guarantees. Finally, we conclude this section with a description of the DECO method for predicting multi-alphabet sequences using binary CTW predictors.

2.1 Tree Sources

The parametric distribution estimated by the CTW algorithm is the set of depth-bounded tree-sources. A tree-source is a variable order Markov model (VMM). Let Σ be an alphabet of size k and D a non-negative integer. A D -bounded tree source is any full k -ary tree¹ whose height $\leq D$. Each leaf of the tree is associated with a probability distribution over Σ . For example, in Figure 1 we depict three tree-sources over a binary alphabet. In this case, the trees are full binary trees. The single node tree in Figure 1(c) is a zero-order (Bernoulli) source and the other two trees (Figure 1(a) and (b)) are 2-bounded sources. Another useful way to view a tree-source is as a set $\mathcal{S} \subseteq \Sigma^{\leq D}$ of “suffixes” in which each $s \in \mathcal{S}$ is a path (of length up to D) from a (unique) leaf to the root. We also refer to \mathcal{S} as the (tree-source) *topology*. For example, $\mathcal{S} = \{0, 01, 11\}$ in Figure 1(b). The path from the middle leaf to the root corresponds to the sequence $s = 01$ and therefore we refer to this leaf simply as s . For convenience we also refer to an internal node by the (unique) path from that node to the root. Observe that this path is a suffix of some $s \in \mathcal{S}$. For example, the right child of the root in Figure 1(b) is denoted by the suffix 1.

The (zero-order) distribution associated with the leaf s is denoted $\mathbf{z}_s(\sigma)$, $\forall \sigma \in \Sigma$, where $\sum_{\sigma} \mathbf{z}_s(\sigma) = 1$ and $\mathbf{z}_s(\cdot) \geq 0$.

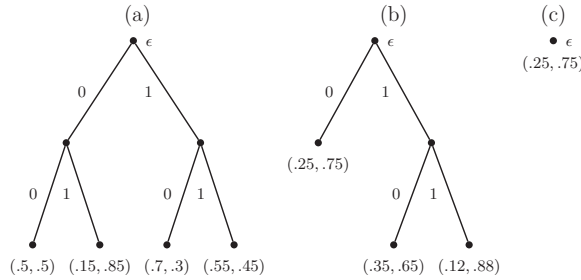


Figure 1: Three examples for $D = 2$ bounded tree-sources over $\Sigma = \{0, 1\}$. The corresponding suffix-sets are $\mathcal{S}_{(a)} = \{00, 10, 01, 11\}$, $\mathcal{S}_{(b)} = \{0, 01, 11\}$, and $\mathcal{S}_{(c)} = \{\epsilon\}$ (ϵ is the empty sequence). The probabilities for generating $\mathbf{x}_1^3 = 100$ given initial context 00 are $P_{(a)}(100|00) = P_{(a)}(1|00)P_{(a)}(0|01)P_{(a)}(0|10) = 0.5 \cdot 0.7 \cdot 0.15$, $P_{(b)}(100|00) = 0.75 \cdot 0.35 \cdot 0.25$, and $P_{(c)}(100|00) = 0.75 \cdot 0.25 \cdot 0.25$.

We denote the set of all D -bounded tree-source topologies (suffix sets) by \mathcal{C}_D . For example, $\mathcal{C}_0 = \{\{\epsilon\}\}$ and $\mathcal{C}_1 = \{\{\epsilon\}, \{0, 1\}\}$, where ϵ is the empty sequence.

For each n , a D -bounded tree-source induces a probability distribution over the set Σ^n of all n -length sequences. This distribution depends on an initial “context” (or “state”), $\mathbf{x}_{1-D}^0 = x_{1-D} \cdots x_0$, which can be any sequence in Σ^D . The tree-source induced probability

1. A full k -ary tree is a tree in which each node has exactly zero or k children.

of the sequence $\mathbf{x}_1^n = x_1x_2 \cdots x_n$ is, by the chain rule,

$$P_{\mathcal{S}}(\mathbf{x}_1^n) = \prod_{t=1}^n P_{\mathcal{S}}(x_t | \mathbf{x}_{t-D+1}^{t-1}), \quad (2)$$

where $P_{\mathcal{S}}(x_t | \mathbf{x}_{t-D}^{t-1})$ is $\mathbf{z}_s(x_t) = P_{\mathcal{S}}(x_t | s)$ and s is the (unique) suffix of \mathbf{x}_{t-D}^{t-1} in \mathcal{S} . Clearly, a tree-source can generate sequences: the i th symbol is randomly drawn using the conditional distribution $P_{\mathcal{S}}(\cdot | \mathbf{x}_{i-D}^{i-1})$. Let $\text{SUB}_s(\mathbf{x}_1^n)$ be the *ordered* non-contiguous sub-sequence of symbols appearing after the context s in \mathbf{x}_1^n . For example, if $\mathbf{x}_1^8 = 01100101$, and $s = 0$, then, $\text{SUB}_s(\mathbf{x}_1^8) = 1011$. Let s be any suffix in \mathcal{S} and $\mathbf{y}_1^m = \text{SUB}_s(\mathbf{x}_1^n)$. For every $\mathbf{x}_1^n \neq \epsilon$ we define $\mathbf{z}_s(\mathbf{x}_1^n) = \prod_{i=1}^m \mathbf{z}_s(y_i)$ and for the empty sequence $\mathbf{z}_s(\epsilon) = 1$. Thus, we can rewrite Equation (2) as

$$P_{\mathcal{S}}(\mathbf{x}_1^n) = \prod_{s \in \mathcal{S}} \mathbf{z}_s(\mathbf{x}_1^n). \quad (3)$$

2.2 The Context-Tree Weighting Method

Here we describe the CTW prediction algorithm (Willems et al., 1995), originally presented as a lossless compression algorithm.² The goal of the CTW algorithm is to predict a sequence (nearly) as good as the the best tree-source. This goal can be divided into two sub-problems. The first is to guess the topology of the best tree-source, and the second is to estimate the distributions associated with its leaves.

Suppose, first, that the best tree topology (i.e., the suffix-set \mathcal{S}) is known. A good solution assigns to each $s \in \mathcal{S}$ a *zero-order estimator* $\hat{\mathbf{z}}_s$ that estimates the true probability distribution \mathbf{z}_s associated with s . This can be done using standard statistical methods; that is, by considering all occurrences of s in \mathbf{x}_1^n and constructing $\hat{\mathbf{z}}_s$ via counting and smoothing. We currently consider $\hat{\mathbf{z}}_s$ as a generic estimator and discuss specific implementations later on.

In practice, however, the best tree-source's topology is unknown. Instead of guessing this topology, CTW considers all possible D -bounded topologies \mathcal{S} (each is a subtree of the perfect k -ary tree), and for each \mathcal{S} it constructs a predictor by estimating its zero-order leaf probabilities. CTW then takes a weighted mixture of all these predictors, corresponding to all topologies. Clearly, there are exponentially many D -bounded topologies. The beauty of the CTW algorithm is the efficient computation of this mixture of exponential size.

In the following description of the CTW algorithm, the output of the algorithm is a probability $P_{\text{CTW}}(\mathbf{x}_1^n)$ for the entire sequence \mathbf{x}_1^n . Observe that this is equivalent to estimating the next-symbol probabilities because

$$P_{\text{CTW}}(\sigma | \mathbf{x}_1^n) = P_{\text{CTW}}(\mathbf{x}_1^n \sigma) / P_{\text{CTW}}(\mathbf{x}_1^n) \quad (4)$$

for each $\sigma \in \Sigma$ (provided that these probabilities can be marginalized, i.e., $\sum_{\sigma} P_{\text{CTW}}(\mathbf{x}_1^n \sigma) = P_{\text{CTW}}(\mathbf{x}_1^n)$).

We require the following definitions. Let \mathbf{x}_1^n be any sequence (in Σ^n) and fix a bound D and an initial context \mathbf{x}_{1-D}^0 . Let s be any context in \mathcal{S} , and $\mathbf{y}_1^m = \text{SUB}_s(\mathbf{x}_1^n)$. The *sequential*

2. As mentioned above, any lossless compression algorithm can be translated into a sequence prediction algorithm and vice versa (see, e.g., Merhav and Feder, 1998).

zero-order estimation for \mathbf{x}_1^n is, by the chain-rule,

$$\hat{z}_s(\mathbf{x}_1^n) = \prod_{i=1}^m \hat{z}(y_i | \mathbf{y}_1^{i-1}), \quad (5)$$

where $\mathbf{y}_1^0 = \epsilon$ and $\hat{z}(y_i | \mathbf{y}_1^{i-1})$ is a zero-order probability estimate based on the symbol counts in \mathbf{y}_1^{i-1} . The product of such predictions is $\hat{z}_s(\mathbf{x}_1^n)$, and hence, we refer to it as a sequential zero-order estimate.

We now describe the main CTW idea via a simple example and then provide a pseudo-code for the general CTW algorithm. Consider a binary alphabet and the case $D = 1$. Here, CTW works on the perfect binary tree of height one and therefore should mix the predictions associated with two topologies: $\mathcal{S}_0 = \{\epsilon\}$ (where ϵ is the empty sequence), and $\mathcal{S}_1 = \{0, 1\}$. Note that \mathcal{S}_0 corresponds to the zero-order topology as in Figure 1(c). The algorithm takes a mixture of the zero-order estimate $\hat{z}_\epsilon(\mathbf{x}_1^n)$ and the one-order estimate. The latter is exactly $\hat{z}_0(\mathbf{x}_1^n) \cdot \hat{z}_1(\mathbf{x}_1^n)$ because \hat{z}_0 and \hat{z}_1 are independent. Thus, the final estimate is

$$P_{\text{CTW}}(\mathbf{x}_1^n) = \frac{1}{2} \hat{z}_\epsilon(\mathbf{x}_1^n) + \frac{1}{2} (\hat{z}_0(\mathbf{x}_1^n) \cdot \hat{z}_1(\mathbf{x}_1^n)).$$

For larger trees ($D > 1$), CTW uses the same idea, but now, instead of taking zero-order estimates for the root's children, the CTW algorithm recursively computes their estimates. The pseudo-code of the CTW recursive mixture computation appears in Algorithm 1. We later show in Lemma 3 that this code calculates the mixture of all D -bounded tree-source predictions weighted by their complexities, which are defined as follows.

Definition 1 Let $T_{\mathcal{S}}$ denote the tree associated with the suffix set \mathcal{S} . The complexity of $T_{\mathcal{S}}$ is defined to be

$$|T_{\mathcal{S}}| = |\{s \in \mathcal{S} : |s| < D\}| + \frac{|\mathcal{S}| - 1}{k - 1}.$$

Recall that the number of leaves in $T_{\mathcal{S}}$ is exactly $|\mathcal{S}|$ and there are $\frac{|\mathcal{S}|-1}{k-1}$ internal nodes in any full k -ary tree. Therefore, $|T_{\mathcal{S}}|$ is the number of nodes in $T_{\mathcal{S}}$ minus the number of leaves $s \in \mathcal{S}$ with maximal depth D .

For example, let $T_{(a)}$ be the tree of Figure 1(a) (resp. for (b) and (c)); $|T_{(a)}| = 0 + 3 = 3$; $|T_{(b)}| = 1 + 2 = 3$ ($= |T_{(a)}$); $|T_{(c)}| = 1 + 0 = 1$.

Observation 2 Let $\mathcal{S}_\sigma = \{s : s\sigma \in \mathcal{S}\}$. For any D -bounded topology \mathcal{S} , $|\mathcal{S}| > 1$,

$$|T_{\mathcal{S}}| = 1 + \sum_{\sigma \in \Sigma} |T_{\mathcal{S}_\sigma}|.$$

Note that \mathcal{S}_σ is a $(D - 1)$ -bounded topology. Note also that the complexity depends on D . Therefore, for the base case (when $|\mathcal{S}| = 1$), the complexity of $T_{\mathcal{S}}$ is zero if $D = 0$ and one if $D \geq 1$.

The proof of the following lemma is a straightforward generalization of the one for binary alphabets by Willems et al. (1995).

Algorithm 1 The context-tree weighting algorithm

/ This code calculates the CTW probability for the (whole) sequence \mathbf{x}_1^n , $P_{\text{CTW}}(\mathbf{x}_1^n | \mathbf{x}_{1-D}^0)$. The input arguments include the sequence \mathbf{x}_1^n , an initial context \mathbf{x}_{1-D}^0 (that determines the suffixes for predicting the first symbols), a bound D on the order, and an implementation for the sequential zero-order estimators $\hat{\mathbf{z}}_s(\cdot)$. The code uses the `mix` procedure (see below). */*

```
CTW( $\mathbf{x}_1^n$ ,  $\mathbf{x}_{1-D}^0$ ,  $D$ ,  $\hat{\mathbf{z}}_s(\cdot)$ ) {  
  for every  $s \in \Sigma^{\leq D}$  do  
    calculate and store  $\hat{\mathbf{z}}_s(\mathbf{x}_1^n)$  as given in Equation (5).  
  end for  
  return  $P_{\text{CTW}}(\mathbf{x}_1^n) = \text{mix}(\epsilon, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0)$ .  
}
```

/ This procedure mixes the predictions of all continuations s 's of $s \in \Sigma^{\leq D}$, such that s 's is also in $\Sigma^{\leq D}$. Note that the context of the first few symbols is determined by the initial context \mathbf{x}_{1-D}^0 . */*

```
mix( $s$ ,  $\mathbf{x}_1^n$ ,  $\mathbf{x}_{1-D}^0$ ) {  
  if  $|s| = D$  then  
    return  $\hat{\mathbf{z}}_s(\mathbf{x}_1^n)$ .  
  else  
    return  $\frac{1}{2} \hat{\mathbf{z}}_s(\mathbf{x}_1^n) + \frac{1}{2} \prod_{\sigma \in \Sigma} \text{mix}(\sigma s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0)$ .  
  end if  
}
```

Lemma 3 Let $0 \leq d - 1 \leq D$ and $s \in \Sigma^{d-1}$. Then,

$$\text{mix}(s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) = \sum_{\mathcal{U} \in \mathcal{C}_{D-d+1}} 2^{-|T_{\mathcal{U}}|} \prod_{u \in \mathcal{U}} \hat{\mathbf{z}}_{us}(\mathbf{x}_1^n).$$

Recall that \mathcal{C}_m is the set of all m -bounded topologies; `mix` is defined in Algorithm 1.

Proof By induction on $D - d$. When $D - d = 0$, $\mathcal{C}_{D-d} = \mathcal{C}_0$ contains only the single-node topology $\mathcal{U} = \{\epsilon\}$. In this case $|T_{\mathcal{U}}| = 0 - \frac{1-1}{k-1} = 0$, by Definition 1. Notice that the size $|s| = d = D$, so $\text{mix}(s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) = \hat{\mathbf{z}}_s(\mathbf{x}_1^n)$. We conclude that,

$$\text{mix}(s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) = \hat{\mathbf{z}}_s(\mathbf{x}_1^n) = 2^{-0 - \frac{1-1}{k-1}} \hat{\mathbf{z}}_s(\mathbf{x}_1^n) = \sum_{\mathcal{U} \in \mathcal{C}_0} 2^{-|T_{\mathcal{U}}|} \prod_{u \in \mathcal{U}} \hat{\mathbf{z}}_{us}(\mathbf{x}_1^n).$$

Assume that the statement holds for some $0 < D - d$ and consider the case $D - d + 1$; that is, $|s| = d - 1 < D$. In this case $\mathcal{U} \in \mathcal{C}_{D-d+1}$. In the following derivations we also refer to alphabet symbols by their indices, $i = 1, \dots, k$ (according to some fixed order) or by σ_i . For example, \mathcal{U}_i is the topology corresponding to the subtree of $T_{\mathcal{U}}$ whose root is defined

by σ_i ; thus, \mathcal{U}_i is a $D - d$ bounded tree-source. We thus have

$$\text{mix}(s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) = \frac{1}{2} \hat{z}_s(\mathbf{x}_1^n) + \frac{1}{2} \prod_{\sigma \in \Sigma} \text{mix}(\sigma s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) \quad (6)$$

$$= \frac{1}{2} \hat{z}_s(\mathbf{x}_1^n) + \frac{1}{2} \prod_{\sigma \in \Sigma} \left\{ \sum_{\mathcal{U} \in \mathcal{C}_{D-d}} 2^{-|\mathcal{U}|} \prod_{u \in \mathcal{U}} \hat{z}_{u\sigma s}(\mathbf{x}_1^n) \right\} \quad (7)$$

$$= \frac{1}{2} \hat{z}_s(\mathbf{x}_1^n) + \sum_{\mathcal{U}_1} \dots \sum_{\mathcal{U}_k} 2^{-(1+\sum_{i=1}^k |\mathcal{U}_i|)} \prod_{u \in \mathcal{U}_1} \hat{z}_{u\sigma_1 s}(\mathbf{x}_1^n) \cdots \prod_{u \in \mathcal{U}_k} \hat{z}_{u\sigma_k s}(\mathbf{x}_1^n) \quad (8)$$

$$= \sum_{\mathcal{U} \in \mathcal{C}_{D-d+1}} 2^{-|\mathcal{U}|} \prod_{u \in \mathcal{U}} \hat{z}_{us}(\mathbf{x}_1^n), \quad (9)$$

where step (6) is by the definition of $\text{mix}(s, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0)$; (7) is by the induction hypothesis; (8) is by exchanging the product of sums with sums of products; and finally, (9) follows from Observation 2. \blacksquare

The next corollary expresses the CTW prediction as a mixture of all D -bounded tree-sources. The proof of this corollary directly follows from Lemma 3 and from the definition of $P_{\text{CTW}}(\mathbf{x}_1^n)$ in Algorithm 1.

Corollary 4

$$P_{\text{CTW}}(\mathbf{x}_1^n) = \text{mix}(\epsilon, \mathbf{x}_1^n, \mathbf{x}_{1-D}^0) = \sum_{S \in \mathcal{C}_D} 2^{-|S|} \prod_{s \in S} \hat{z}_s(\mathbf{x}_1^n). \quad (10)$$

Remark 5 *The number of tree-source topologies in \mathcal{C}_D is superexponential (recall that each $S \in \mathcal{C}$ is a pruning of the perfect k -ary tree of height D). Thus, for practical reasons, the calculation of Equation (10) must be efficient. The pseudo-code of the CTW in Algorithm 1 is conceptual rather than efficient. However, the beauty of the CTW is that it can calculate the tree-source mixture in linear time with respect to n . For a description of an efficient implementation of the CTW algorithm, see for example, Sadakane et al. (2000) and Chapter 4.4 of Volf (2002). Our Java implementation of the CTW algorithm can be found at <http://www.cs.technion.ac.il/~rani/code/umm>.*

2.3 Analysis of CTW for Multi-Alphabets

The analysis of CTW for multi-alphabets (multi-CTW) relies upon specific implementations of the sequential zero-order estimators $\hat{z}_s(\cdot)$. Such estimators are in general counters of past events. However, these estimators should not neglect unobserved events. In the context of log-loss prediction, assigning zero probability to these “zero frequency” events is harmful because the log-loss of an unobserved but possible event is infinite. The problem of assigning probability mass to unobserved events is also called the “missing-mass problem” (or the “zero frequency problem”).

The original CTW algorithm applies the well-known KT estimator (Krichevsky and Trofimov, 1981).

Definition 6 Fix any \mathbf{x}_1^n and let N_σ be the frequency of $\sigma \in \Sigma$ in \mathbf{x}_1^n . The KT estimator assigns the following (sequential zero-order) probability to the sequence \mathbf{x}_1^n ,

$$\hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n) = \hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^{n-1}) \frac{N_{x_n} + 1/2}{\sum_{\sigma \in \Sigma} N_\sigma + k/2}, \quad (11)$$

where $\hat{\mathbf{z}}_s^{\text{KT}}(\epsilon) = 1$.

Observe that the term $P(\sigma|\mathbf{x}_1^n) = \frac{N_\sigma + 1/2}{\sum_{\sigma \in \Sigma} N_\sigma + k/2}$, is an *add-half* predictor that belongs to the family of add-constant predictors.³

The KT estimator provides a prediction that is uniformly close to the set \mathcal{Z} of zero-order distributions over Σ . Each distribution $\mathbf{z} \in \mathcal{Z}$ is a probability vector from $(\mathbb{R}_+)^k$, and $\mathbf{z}(\sigma)$ denotes the probability of σ . Thus, $\mathbf{z}(\mathbf{x}_1^n) = \prod_{\sigma} \mathbf{z}(\sigma)^{N_\sigma}$. The next theorem provides a performance guarantee on the worst-case redundancy of the KT estimator. This guarantee is for a whole sequence \mathbf{x}_1^n . Notice that the per-symbol redundancy of KT diminishes with n at a rate $\frac{\log n}{n}$. For completeness, the proof of the following theorem is provided in Appendix A.

Theorem 7 (Krichevsky and Trofimov) Let Σ be any alphabet with $|\Sigma| = k \geq 2$. For any sequence $\mathbf{x}_1^n \in \Sigma^n$,

$$R_{\text{KT}}(\mathbf{x}_1^n, \mathbf{z}) = \log \sup_{\mathbf{z} \in \mathcal{Z}} \mathbf{z}(\mathbf{x}_1^n) - \log \hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n) \leq \frac{k-1}{2} \log n + \log k. \quad (12)$$

Remark 8 Krichevsky and Trofimov (1981) originally defined KT to be a mixture of all zero-order distributions in \mathcal{Z} , weighted by the Dirichlet (1/2) distribution. Thus, this mixture is

$$\hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n) = \int_{\mathcal{Z}} w(d\mathbf{z}) \mathbf{z}(\mathbf{x}_1^n),$$

where $w(d\mathbf{z})$ is the Dirichlet distribution with parameter 1/2 defined by

$$w(d\mathbf{z}) = \frac{1}{\sqrt{k}} \frac{\Gamma(\frac{k}{2})}{\Gamma(\frac{1}{2})^k} \prod_{i=1}^k z(i)^{-1/2} \lambda(dz), \quad (13)$$

$\Gamma(x) = \int_{\mathbb{R}^+} t^{x-1} \exp(-t) dt$ is the gamma function (see, for example, Courant and John, 1989), and $\lambda(\cdot)$ is a measure on \mathcal{Z} . Shtarkov (1987) was the first to show that this mixture can be calculated sequentially as in Definition 6.

The upper bound of Theorem 7 on the redundancy of the KT estimator is a key element in the proof of the following theorem, providing a finite-sample point-wise redundancy bound for the multi-CTW (see, e.g., Tjalkens et al., 1993; Catoni, 2004).

Theorem 9 (Willems et al.) Let Σ be any alphabet with $|\Sigma| = k \geq 2$. For any sequence $\mathbf{x}_1^n \in \Sigma^n$ and any D -bounded tree-source with a topology \mathcal{S} and distribution $P_{\mathcal{S}}$, the following holds:

$$R_{\text{CTW}}(\mathbf{x}_1^n, P_{\mathcal{S}}) \leq \begin{cases} n \log k + \frac{k|\mathcal{S}|-1}{k-1}, & n < |\mathcal{S}|; \\ \frac{(k-1)|\mathcal{S}|}{2} \log \frac{n}{|\mathcal{S}|} + |\mathcal{S}| \log k + \frac{k|\mathcal{S}|-1}{k-1}, & n \geq |\mathcal{S}|. \end{cases}$$

3. Another famous add-constant predictor is the add-one predictor, also called *Laplace's law of succession* (Laplace, 1995).

Proof

$$\begin{aligned}
R_{\text{CTW}}(\mathbf{x}_1^n, P_{\mathcal{S}}) &= \log P_{\mathcal{S}}(\mathbf{x}_1^n) - \log P_{\text{CTW}}(\mathbf{x}_1^n) \\
&= \underbrace{\log \frac{P_{\mathcal{S}}(\mathbf{x}_1^n)}{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}}_{(i)} + \underbrace{\log \frac{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}{P_{\text{CTW}}(\mathbf{x}_1^n)}}_{(ii)}
\end{aligned} \tag{14}$$

We now bound the term (14)(i) and define the following auxiliary function:

$$f(x) = \begin{cases} x \log k & , 0 \leq x < 1; \\ \frac{k-1}{2} \log x + \log k & , x \geq 1. \end{cases}$$

Note that this function is continuous and concave in $[0, \infty)$. Let $N_{\sigma}(s)$ denote the frequency of σ in $\text{SUB}_s(\mathbf{x}_1^n)$. Thus,

$$\log \frac{P_{\mathcal{S}}(\mathbf{x}_1^n)}{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)} = \sum_{s \in \mathcal{S}} \log \frac{\mathbf{z}_s(\mathbf{x}_1^n)}{\hat{z}_s(\mathbf{x}_1^n)} \tag{15}$$

$$\leq \sum_{\substack{s \in \mathcal{S}, \text{ s.t.} \\ N_{\sigma}(s) > 0}} \left(\frac{k-1}{2} \log \left(\sum_{\sigma} N_{\sigma}(s) \right) + \log k \right) \tag{16}$$

$$\begin{aligned}
&= |\mathcal{S}| \sum_{s \in \mathcal{S}} \frac{1}{|\mathcal{S}|} f \left(\sum_{\sigma} N_{\sigma}(s) \right) \\
&\leq |\mathcal{S}| f \left(\frac{\sum_{s \in \mathcal{S}} \sum_{\sigma} N_{\sigma}(s)}{|\mathcal{S}|} \right)
\end{aligned} \tag{17}$$

$$\begin{aligned}
&= |\mathcal{S}| f \left(\frac{n}{|\mathcal{S}|} \right) \\
&= \begin{cases} n \log k, & n < |\mathcal{S}|; \\ \frac{(k-1)|\mathcal{S}|}{2} \log \frac{n}{|\mathcal{S}|} + |\mathcal{S}| \log k, & n \geq |\mathcal{S}|, \end{cases}
\end{aligned} \tag{18}$$

where step (15) follows from an application of Equation (3); step (16) is by the performance guarantee for the KT prediction, as given in Theorem 7; and step (17) is by Jensen's inequality.

We now bound the term (14)(ii)

$$\log \frac{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}{P_{\text{CTW}}(\mathbf{x}_1^n)} = \log \frac{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}{\sum_{\mathcal{S} \in \mathcal{C}_D} 2^{-|\mathcal{S}|} \prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)} \tag{19}$$

$$\leq \log \frac{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}{\sum_{\mathcal{S} \in \mathcal{C}_D} 2^{-\frac{k|\mathcal{S}|-1}{k-1}} \prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)} \tag{20}$$

$$\begin{aligned}
&\leq \log \frac{\prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)}{2^{-\frac{k|\mathcal{S}|-1}{k-1}} \prod_{s \in \mathcal{S}} \hat{z}_s(\mathbf{x}_1^n)} \\
&= \log 2^{\frac{k|\mathcal{S}|-1}{k-1}} \\
&= \frac{k|\mathcal{S}| - 1}{k - 1},
\end{aligned} \tag{21}$$

where in step (19) we applied Equation (10) and the justification for (20) is that $|\{s \in \mathcal{S} : |s| < D\}| \leq |\mathcal{S}|$. Thus, according to Definition 1, $|T_{\mathcal{S}}| \leq |\mathcal{S}| + \frac{|\mathcal{S}|-1}{k-1} = \frac{k|\mathcal{S}|-1}{k-1}$. We complete the proof by summing up (18) and (21). \blacksquare

Remark 10 *The CTW bound used by Catoni (2004) is somewhat tighter than the bound of Theorem 9 but contains some implicit terms.*

Remark 11 *Willems (1998) provided extensions for the CTW algorithm that eliminate its dependency on the maximal bound D and the initial context \mathbf{x}_{1-D}^0 . For the extended algorithm and binary prediction problems, Willems derived a point-wise redundancy bound of*

$$\frac{|\mathcal{S}|}{2} \log \frac{n - \Delta_s(\mathbf{x}_1^n)}{|\mathcal{S}|} + 2|\mathcal{S}| - 1 + \Delta_s(\mathbf{x}_1^n),$$

where $\Delta_s(\mathbf{x}_1^n) \leq D$ denotes the number of symbols in the prefix of \mathbf{x}_1^n that do not appear after a suffix $s \in \mathcal{S}$.

Remark 12 *Interestingly, it can be shown that the CTW algorithm is an instance of the well-known generic expert-advice algorithm of Vovk (1990). This observation is new, to the best of our knowledge, although there are citations that connect the CTW algorithm with the expert advice scheme (see, e.g., Merhav and Feder, 1998; Helmbold and Schapire, 1997).*

It can be shown that these two algorithms are identical when Vovk’s algorithm is applied with the log-loss (see, e.g., Haussler et al., 1998, example 3.12). In this case, the set of experts in Vovk’s algorithm consists of all D -bounded tree-sources, \mathcal{C}_D ; the initial weight of each expert, \mathcal{S} , corresponds to its complexity $|T_{\mathcal{S}}|$; and the weight of each expert at round t equals $2^{-|T_{\mathcal{S}}|} P_{\mathcal{S}}(\mathbf{x}_1^{t-1})$. Note, however, that the power of the CTW method is in its efficiency in mixing exponentially many sources (or experts). Vovk’s algorithm is not concerned with how to compute this average.

2.4 Hierarchical CTW Decompositions

The CTW algorithm is known to achieve excellent empirical performance in *binary* prediction problems. However, when applying CTW on sequences over larger alphabets, the resulting performance falls short of the best known performance (Tjalkens et al., 1997). This fact motivates different approaches for applying the CTW algorithm on multi-alphabet sequences. Volf targeted this issue in his Ph.D. thesis (2002). Following Tjalkens et al. (1994), who proposed a rudimentary alphabet decomposition approach, he studied a solution to the multi-alphabet prediction problem that is based on a tree hierarchy of binary problems. Each of these binary problems is solved using a slight variation of the binary CTW algorithm. We now describe the resulting ‘decomposed CTW’ approach, which we term for short the “DECO” algorithm.

Consider a full binary *decomposition tree* T with $k = |\Sigma|$ leaves, where each leaf is uniquely associated with a symbol in Σ . Each internal node v of T corresponds to the binary problem of predicting whether the next symbol is a leaf on v ’s left subtree or a leaf on v ’s right subtree. For example, for $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{r}\}$, Figure 2 depicts a decomposition

There are many possibilities for constructing the decomposition tree T .⁴ A major open problem is how to identify useful decomposition trees. Intuitively, it appears that placing high frequency symbols close to the root is a good idea for two reasons: (i) When traversing the tree from the root to such symbols, the number of visits to other internal nodes is minimized, thus reducing extra loss; (ii) High frequency symbols appearing closer to the root could be involved in “easier” binary problems because of the denser statistics we have on them.

Tjalkens et al. (1997) and Volf (2002, Chapter 5) suggested taking T as the Huffman coding tree computed with respect to the frequency counts of the symbols in \mathbf{x}_1^n . While intuitively appealing, there is currently no compelling explanation for this heuristic. In Section 3.1 we provide a formal motivation for Huffman decompositions.

3. Redundancy Bounds For the DECO Algorithm

We start this section with some definitions that formalize the hierarchical alphabet decomposition approach. We also define a new category of sources called “decomposed sources,” which will aid in the analysis of algorithm DECO. To this end, we use an equivalence between decomposed sources and the ordinary tree-sources of Section 2.1. The main result of this section is Theorem 19, providing a pointwise redundancy bound for the DECO algorithm. This bound, in particular, implies a performance guarantee for Huffman decomposition trees, which is given in Corollary 23.

Let Σ be a multi-alphabet with k symbols and fix some order bound D and initial context \mathbf{x}_{1-D}^0 . We refer to a decomposition-tree (see Section 2.4) simply as a *tree* and to an ordinary tree source as a *multi-source*, denoted by $M = (\mathcal{S}, P_{\mathcal{S}})$.

Definition 13 (Decomposed Source) *A (D -bounded) decomposed source \mathcal{T} over Σ is a pair*

$$\mathcal{T} = (T, \{M_1, M_2, \dots, M_{k-1}\}),$$

where T is a (decomposition) tree over Σ and for each internal node, $v \in T$, there is a matching source $M_v = (\mathcal{S}_v, P_v)$ whose suffix set, \mathcal{S}_v , contains all paths of some full k -ary tree (of maximal height D). Additionally, for every $s \in \mathcal{S}_v$, $P_v(\cdot|s)$ is a binary distribution over $\{0_v, 1_v\}$. Note that M_i is not a standard multi-source because it predicts binary sequences of supersymbols while depending on multi-alphabet contexts. Such sources will always be denoted by M_v for some internal node v . Let $\mathbf{x} \in \Sigma^D$ be any sequence and $\sigma \in \Sigma$. The prediction induced by \mathcal{T} is

$$P_{\mathcal{T}}(\sigma|\mathbf{x}) = \prod_{v, \text{ s.t.}, \sigma \in \Sigma_v} P_v(\sigma|\mathbf{x}). \quad (22)$$

4. We can map every decomposition tree with the partition of 1 into sums of k terms, each of which is a power of $1/2$, where each leaf σ at level ℓ_σ defines the power $(1/2)^{\ell_\sigma}$. (This is possible due to Kraft’s inequality.) Therefore, the number of such decomposition trees is obtained by multiplying $k!$ (all permutations of Σ) with this number of partitions. The former is known as sequence A002572 in Sloane and Plouffe (1995). For example, for $k = 26$ we have $26! \cdot 565168 = 227927428502001453851738112000000$ possible decomposition trees.

We say that two probabilistic tree-sources over Σ are *equivalent* if they agree on the probability of every sequence $\mathbf{x} \in \Sigma^*$. Note that two structurally different tree-sources can be equivalent. A multi-source is *minimal* if it has no redundant suffixes. A decomposed source is minimal if all its M_v models are minimal. The formal definitions follow.

Definition 14 (Minimal Sources) (i) A multi-source $M = (\mathcal{S}, P_{\mathcal{S}})$ is *minimal* if there is no $s \in \Sigma^{<D}$ for which $P_{\mathcal{S}}(\cdot|\sigma_i s) = P_{\mathcal{S}}(\cdot|\sigma_j s)$ for all $\sigma_i \neq \sigma_j$ and both $\sigma_i s$ and $\sigma_j s$ are in \mathcal{S} . (ii) We say that $\mathcal{T} = (T, \{M_v\})$ is a *minimal decomposed source* if for all internal nodes v of T , M_v is minimal.

For example, we depict in Figure 3 two equivalent multi-sources. The multi-source in Figure 3 (a) is a minimal multi-source while the multi-source in Figure 3 (b) is not minimal.

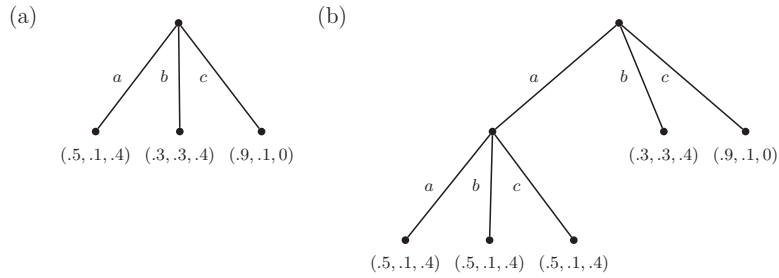


Figure 3: An example of two equivalent multi-sources. Both sources generate the same probability to every sequence of length larger than two. Take for example $\mathbf{x} = \mathbf{aaba}$ with initial context \mathbf{ba} . Both sources will induce the following prediction: $P(\mathbf{aaba}|\mathbf{ba}) = 0.5 \cdot 0.5 \cdot 0.1 \cdot 0.3 = 0.0075$. Observe that the source in (a) is minimal while the other source is not.

There is a simple procedure for transforming a non-minimal source into its equivalent minimal form: Replace each redundant suffix, σs , with its suffix s . That is, trim all children of s and assign $P_{\mathcal{S}}(\cdot|s) = P_{\mathcal{S}}(\cdot|\sigma s)$ for some σ .

The following two lemmas facilitate a “translation” between decomposed and multi sources.

Lemma 15 For every multi-source M and tree T there exists a minimal decomposed source $\mathcal{T} = (T, \{M_v\})$ such that M and \mathcal{T} are equivalent.

Proof Let $M = (\mathcal{S}, P)$ be a D -bounded multi-source and let T be a tree. We start with the definition of the models $M_v = (\mathcal{S}_v, P_v)$ and set the suffix set $\mathcal{S}_v = \mathcal{S}$, for every internal node v . Let $P_{\mathcal{S}}(0_v|s) = \sum_{\sigma \in 0_v} P_{\mathcal{S}}(\sigma|s)$ for any (internal node) v and $s \in \mathcal{S}_v (= \mathcal{S})$. Similarly, $P_{\mathcal{S}}(1_v|s) = \sum_{\sigma \in 1_v} P_{\mathcal{S}}(\sigma|s)$. Let $\text{parent}(v)$ denote the parent of node v . For every internal node v and $s \in \mathcal{S}_v$ we define

$$P_v(0_v|s) = P_{\mathcal{S}}(0_v|s) / P_{\text{parent}(v)}(0_v \cup 1_v|s),$$

and similarly for $P_v(1_v|s)$. For the base case (i.e., the root) we do not divide by the denominator. Clearly, $P_v(\cdot|s)$ is a valid distribution and the resulting structure $\mathcal{T} = (T, \{M_v\})$ is a decomposed source.

We shall now prove that \mathcal{T} and M are equivalent. Recall that $\mathcal{S} = \mathcal{S}_v$ for every internal node v . Let v be any internal node in T and $u = \text{parent}(v)$. Assume, without loss of generality, that $0_v \subset 1_u$, and therefore, $0_v \cup 1_v = 1_u$. Note that, for every $s \in \mathcal{S}$,

$$P_u(1_u|s)P_v(0_v|s) = P_u(1_u|s) (P_{\mathcal{S}}(0_v|s)/P_u(0_v \cup 1_v|s)) = P_{\mathcal{S}}(0_v|s).$$

Therefore, $P_{\mathcal{T}}(\sigma|s)$ of Equation (22) is a telescopic product; hence, for every $\sigma \in \Sigma$ and $s \in \mathcal{S}$, $P_{\mathcal{T}}(\sigma|s) = P_{\mathcal{S}}(\sigma|s)$. This proves that M and \mathcal{T} are equivalent. Finally, for minimality, we replace every \mathcal{S}_v with its minimal source. \blacksquare

Lemma 16 *For every decomposed source \mathcal{T} there exists a minimal multi-source M that is equivalent to \mathcal{T} .*

Proof Let $\mathcal{T} = (T, \{M_v = \{\mathcal{S}_v, P_v\}\})$ be a decomposed source. We provide the following constructive scheme for building the equivalent multi-source, $M = (\mathcal{S}, P_{\mathcal{S}})$. Start with $M = M_{\text{root}}$ (the model corresponding to the root of T). We traverse the internal vertices of T (minus the root) such that parent nodes are visited before their descendants (i.e., using preorder). We start with one of the root's children and for each internal node in T we do the following. For each (internal node) $v \in T$ and for every $s_v \in \mathcal{S}_v$, exactly one of the following three cases holds (because both \mathcal{S}_v and \mathcal{S} form a full k -ary tree): (a) $s_v = s \in \mathcal{S}$; or (b) $\exists s \in \mathcal{S}$ such that s is a suffix of s_v ; or (c) $\exists s \in \mathcal{S}$ such that s_v is a suffix of s . We treat these cases as follows. For the first case, we refine the support set of $P_{\mathcal{S}}(\cdot|s)$ by replacing the supersymbol corresponding to $0_v \cup 1_v$ with the two (super) symbols 0_v and 1_v , and define,

$$\begin{aligned} P_{\mathcal{S}}(0_v|s) &= P_{\mathcal{S}}(0_v \cup 1_v|s) \cdot P_v(0_v|s); \\ P_{\mathcal{S}}(1_v|s) &= P_{\mathcal{S}}(0_v \cup 1_v|s) \cdot P_v(1_v|s). \end{aligned} \tag{23}$$

Note that $P_{\mathcal{S}}(0_v \cup 1_v|s)$ has been already assigned (due to the preorder node traversal). Cases (b) and (c) are treated in exactly the same manner. In case (b) also replace s with its extension s_v .

We should now prove that the resulting $M = (\mathcal{S}, P_{\mathcal{S}})$ is a multi-source and that M is equivalent to \mathcal{T} . Both proofs are by induction on $|\Sigma| = k$. For $k = 2$, \mathcal{T} consists only of M_{root} , which is a binary tree source. Hence, obviously, $M = M_{\text{root}}$ is a tree source equivalent to \mathcal{T} . Assume the statement holds for $k - 1 > 2$ and examine $|\Sigma| = k$. Let $v \in T$ be the last visited node in the constructive scheme. Clearly, by the preorder traversal, the children of v are both leaves (both 0_v and 1_v are singletons). Merge the two symbols in $\Sigma_v \subseteq \Sigma$ into some supersymbol σ_v and consider $\mathcal{T}' = (T', \{M_{v'}\})$, which is the decomposed source induced by this replacement. The number of leaves of \mathcal{T}' , which can be denoted $\Sigma' = \Sigma \setminus \Sigma_v \cup \{\sigma_v\}$, is equal to $k - 1$. Thus, by the inductive hypothesis, we construct $M' = \{\mathcal{S}', P_{\mathcal{S}'}\}$, a multi-source that is equivalent to \mathcal{T}' . We now apply the constructive step on M' and v , resulting with $M = (\mathcal{S}, P_{\mathcal{S}})$. Case (b) of the constructive scheme is the only place that we change

\mathcal{S}' (to retrieve \mathcal{S}). \mathcal{S}' is a tree source topology by the induction hypothesis; so is \mathcal{S}_v and clearly, the treatment of case (b) induces a valid tree-source topology (that corresponds to a full k -ary tree). Therefore, \mathcal{S} is a tree-source topology. It is also easy to see that the refinement of the support set of \mathcal{S}' , as in (23), induces a valid distribution over Σ . We conclude that $M = (\mathcal{S}, P_{\mathcal{S}})$ is a multi-source over Σ .

We now turn to prove the equivalence. For every $s \in \mathcal{S}$ and any symbol $\sigma \in \Sigma \setminus \Sigma_v$, we have by Equation (22) that $P_{\mathcal{T}}(\sigma|s) = P_{\mathcal{T}'}(\sigma|s)$, and by the induction hypothesis, $P_{\mathcal{T}'}(\sigma|s) = P_{\mathcal{S}'}(\sigma|s)$. Note that, by the construction, every $s' \in \mathcal{S}'$ is a *suffix* of some $s \in \mathcal{S}$. Therefore, for symbols $\sigma \in \Sigma \setminus \Sigma_v$, $P_{\mathcal{S}'}(\sigma|s') = P_{\mathcal{S}'}(\sigma|s) = P_{\mathcal{S}}(\sigma|s)$ (where s' is the suffix of s). Now for symbols $\sigma \in \Sigma_v$, recall that $|\Sigma_v| = 2$ and therefore, 0_v represents some (ordinary) symbol $\sigma \in \Sigma$ (resp., 1_v). Thus,

$$P_{\mathcal{S}}(\sigma|s) = P_{\mathcal{S}'}(\sigma_v|s)P_v(\sigma|s) \quad (24)$$

$$= P_{\mathcal{T}'}(\sigma_v|s)P_v(\sigma|s) \quad (25)$$

$$= \left(\prod_{u, \text{ s.t., } u \in T' \wedge \sigma \in \Sigma_u} P_u(\sigma|s) \right) P_v(\sigma|s) \quad (26)$$

$$= \prod_{u, \text{ s.t., } u \in T \wedge \sigma \in \Sigma_u} P_u(\sigma|s) \quad (27)$$

$$= P_{\mathcal{T}}(\sigma|s),$$

where (24) is by the construction (23) with $\sigma \in \{0_v, 1_v\}$; (25) is by the induction hypothesis; (26) and (27) are by Equation (22). This proves that M is equivalent to \mathcal{T} . Finally, for satisfying the minimality of M , we take its equivalent minimal multi-source. ■

Remark 17 *It can be shown that a minimal decomposed source (resp., multi-source) is unique. Hence, Lemmas 15 and 16 imply that, for a given tree T , there is a one-to-one mapping between the minimal decomposed sources and multi-sources.*

Consider algorithm DECO applied with a tree T_{DECO} . The redundancy of the DECO algorithm on a sequence \mathbf{x}_1^n , with respect to any decomposed source $\mathcal{T} = (T, \{M_v\})$, is

$$R_{\text{DECO}}(\mathbf{x}_1^n, \mathcal{T}) = \log P_{\mathcal{T}}(\mathbf{x}_1^n) - \log P_{\text{DECO}}(\mathbf{x}_1^n).$$

We do not know how to express this redundancy directly in terms of the unknown source \mathcal{T} . However, we can express it in terms of an equivalent decomposed source \mathcal{T}' that has the same tree as in the algorithm. This “translation” is done using an equivalent multi-source mediator that can be constructed according to Lemmas 15 and 16. To facilitate this discussion, we define, for a decomposed source $\mathcal{T} = (T, \{M_v\})$, its *T' -equivalent* source to be any equivalent decomposition source with tree T' . By Lemmas 15 and 16 this source exists.

Corollary 18 *For any decomposed source $\mathcal{T} = (T, \{M_v\})$ and a tree T' there exists a T' -equivalent source $\mathcal{T}' = (T', \{M'_i\})$.*

Theorem 19 Let T_{DECO} be any tree and \mathbf{x}_1^n a sequence. For every internal node $v \in T_{\text{DECO}}$, denote by CTW_v the corresponding CTW predictor of the DECO algorithm applied with T_{DECO} . Let $\mathcal{T} = (T, \{M_v\})$ be any decomposed source. Then, $R_{\text{DECO}}(\mathbf{x}_1^n, P_{\mathcal{T}}) \leq \sum_{i=1}^{k-1} R_i(\mathbf{x}_1^n)$, where i is an internal-node in T_{DECO} , and

$$R_i(\mathbf{x}_1^n) = \begin{cases} \frac{|\mathcal{S}_i|}{2} \log \frac{n_i}{|\mathcal{S}_i|} + |\mathcal{S}_i| + \frac{k|\mathcal{S}_i|-1}{k-1} & , n_i \geq |\mathcal{S}_i|; \\ n_i + \frac{k|\mathcal{S}_i|-1}{k-1} & , 0 < n_i < |\mathcal{S}_i|; \\ 0 & , n_i = 0. \end{cases} \quad (28)$$

\mathcal{S}_i is the suffix set of the i th (internal) node of the T' -equivalent source of \mathcal{T} , and n_i is the number of times this node is visited when predicting \mathbf{x}_1^n .

Proof Let $\mathcal{T}' = (T_{\text{DECO}}, \{M_{v'}\})$ be the T_{DECO} -equivalent decomposed source of \mathcal{T} . Fix any order on the internal nodes of T' . We will refer to internal nodes both by their order's index and by the notation v . By the chain-rule, $P_v(\mathbf{x}_1^n) = \prod_{x_t \in \Sigma_v} P_v(x_t | x_{1-D}^{t-1})$, where $P_v(x_t | x_{1-D}^{t-1}) = P_v(x_t | s)$ and $s \in \mathcal{S}_v$ is a suffix of x_{1-D}^{t-1} . Thus,

$$\begin{aligned} P_{\mathcal{T}}(\mathbf{x}_1^n) &= P_{\mathcal{T}'}(\mathbf{x}_1^n) & (29) \\ &= \prod_{t=1}^n P_{\mathcal{T}'}(x_t | x_{1-D}^{t-1}) \\ &= \prod_{t=1}^n \prod_{v \in T_{\text{DECO}}, \text{ s.t., } x_t \in \Sigma_v} P_v(x_t | x_{1-D}^{t-1}) \\ &= \prod_{v \in T_{\text{DECO}}} \prod_{x_t \in \Sigma_v} P_v(x_t | x_{1-D}^{t-1}) = \prod_{v \in T_{\text{DECO}}} P_v(\mathbf{x}_1^n), & (30) \end{aligned}$$

where (29) follows from by Corollary 18.

We show that $R_{\text{DECO}}(\mathbf{x}_1^n, P_{\mathcal{T}}) \leq \sum_{i=1}^{k-1} R_i(\mathbf{x}_1^n)$.

$$R_{\text{DECO}}(\mathbf{x}_1^n | P_{\mathcal{T}}) = \log P_{\mathcal{T}}(\mathbf{x}_1^n) - \log P_{\text{DECO}}(\mathbf{x}_1^n) \quad (31)$$

$$= \log P_{\mathcal{T}'}(\mathbf{x}_1^n) - \log P_{\text{DECO}}(\mathbf{x}_1^n) \quad (32)$$

$$= \sum_{j=1}^{k-1} \log P_j(\mathbf{x}_1^n) - \sum_{i=1}^{k-1} \log P_{\text{CTW}_i}(\mathbf{x}_1^n) \quad (33)$$

$$= \sum_{i=1}^{k-1} (\log P_i(\mathbf{x}_1^n) - \log P_{\text{CTW}_i}(\mathbf{x}_1^n)) \quad (34)$$

$$\leq \sum_{i=1}^{k-1} R_i(\mathbf{x}_1^n),$$

where (31) follows from Corollary 18; in Equations (32) and (33) the probabilities P_j and P_i refer to internal nodes of \mathcal{T}' ; in (32) we used Equation (30); and finally, equality (34) directly follows from the proof of Theorem 9. In that proof, we applied the bound (18) for the term (14 i) with $k = 2$, because the zero-order predictors, $z_s(\cdot)$, of CTW_v provide binary predictions. The bound on the term (14 ii) remains as is because CTW_v uses a k -ary tree.

■

The precise values of the model orders $|\mathcal{S}_i|$ in the above upper bound are unknown since the decomposed source is unknown. Nevertheless, for each i , $|\mathcal{S}_i| \leq k^D$. It follows that any DECO scheme is universal with respect to the class of D -bounded (multi) tree-sources. Specifically, given any multi-source, consider its T_{DECO} -equivalent decomposed source \mathcal{T} . For a sequence \mathbf{x}_1^n , by Theorem 19 the per-symbol redundancy is $\frac{1}{n} R_{\text{DECO}}(\mathbf{x}_1^n, P_{\mathcal{T}}) \leq \frac{1}{n} \sum_{i=1}^{k-1} R_i(\mathbf{x}_1^n)$, which vanishes with n since $n_i \leq n$ for every internal-node i .

Remark 20 *The dependency of the DECO algorithm on the maximal bound D and the initial context \mathbf{x}_{1-D}^0 can be eliminated by using the extensions for the CTW algorithm suggested by Willems (1998). Recall that Willems provided a point-wise redundancy bound for this case (see Remark 11). Thus, we can straightforwardly use this result to derive a corresponding bound for the DECO algorithm (the details are omitted).*

3.1 Huffman Decompositions

The general bound of Theorem 19 holds for any decomposition tree. However, it is expected that some trees will result in a tighter bound. Therefore, it is desirable to optimize the bound over all trees. Unfortunately, the sizes $|\mathcal{S}_i|$ are unknown. Even if the sizes $|\mathcal{S}_i|$ were known, it is an NP-hard problem even to decide on the optimal partition corresponding to the root. This hardness result can be obtained by a reduction from MAX-CUT (see, e.g., Papadimitriou, 1994, Chapter 9.3). Hence, we can only hope to approximate the optimal tree.

However, if we replace each $|\mathcal{S}_i|$ value with its maximal value k^D , we are able to show that the bound is optimized when the decomposition tree is the Huffman decoding tree (see, e.g., Cover and Thomas, 1991, Chapter 5.6) of the sequence \mathbf{x}_1^n .

For any decomposition tree T and a sequence \mathbf{x}_1^n , let n_i be the number of times that the inner-node $i \in T$ is visited when predicting \mathbf{x}_1^n using the DECO algorithm. These are precisely the n_i used in Theorem 19, Equation (28). We call these n_i “the counters of T ”.

Lemma 21 *Let \mathbf{x}_1^n be a sequence and T a decomposition tree constructed using Huffman’s procedure, which is based on the empirical distribution $\hat{P}(\sigma) = N_\sigma/n$. Let $\{n_i\}$ be the counters of T . Then, $\sum_{i=1}^{k-1} n_i$ and $\prod_{i=1}^{k-1} n_i$ are both minimal with respect to any other decomposition tree.*

Proof Any tree T induces the following prefix-code over Σ . The codeword of a symbol $\sigma \in \Sigma$ is the path from the root of T to the leaf σ . The length of this code for some T , with respect to \mathbf{x}_1^n , is $\ell(\mathbf{x}_1^n) = \sum_{t=1}^n \ell(x_t)$, where $\ell(x_t)$ is the codeword length of the symbol x_t . It is not hard to see that

$$\ell(\mathbf{x}_1^n) = \sum_{\sigma} N_{\sigma} \cdot \ell(\sigma) = \sum_{i=1}^{k-1} n_i. \quad (35)$$

If T is constructed using Huffman’s algorithm, the average code length, $\frac{1}{n} \sum_{\sigma} N_{\sigma} \cdot \ell(\sigma)$, is the smallest possible. Therefore, T minimizes $\frac{1}{n} \sum_{i=1}^{k-1} n_i$.

To prove that Huffman’s tree also minimizes $\prod_{i=1}^{k-1} n_i$, we define the following lexicographic order on the set of inner nodes of any tree. Given a tree, we let n_v be the counter corresponding to inner node v . We can order the inner nodes, first in ascending order of their counters n_v , and then (among nodes with equal counters), in ascending order of the heights of the sub-trees they root. Let T be a Huffman tree, and T' be any other tree. Let $\{n_v\}$ be the counters of T and let $\{n_{v'}\}$ be the counters of T' . We already know that $\sum_v n_v \leq \sum_{v'} n_{v'}$. We can order (separately) both sets of counters according to the above lexicographic order such that $n_{v_1} \leq \dots \leq n_{v_{k-1}}$ (and similarly, for v'_i). We prove, by induction on k , that $n_{v_i} \leq n_{v'_i}$, for $i = 1, \dots, k - 1$. For $k = 2$ the statement trivially holds. Assume that for $i = 1, \dots, k - 1$, $n_{v_i} \leq n_{v'_i}$. We examine now the case where $i = 1, \dots, k$. According to the construction scheme of the Huffman tree (see, Cover and Thomas, 1991, Chapter 5.6), we have that $n_{v_1} \leq n_{v'_1}$. Note that the children of v_1 and v'_1 are all leaves. Otherwise, the non-leaf child must have the same counter as its parent and is rooting a sub-tree with smaller height. Therefore, by our lexicographic order, the counter of this child must appear before the counter of its parent, which is a contradiction. Thus, if we replace v_1 (resp., v'_1) with a leaf, we do not change the counter’s order for the other nodes for which the inductive hypothesis holds. ■

Remark 22 *After establishing Lemma 21, we found that Glassey and Karp (1976) showed that if $f(\cdot)$ is an arbitrary concave function, then the Huffman tree minimizes $\sum_{i=1}^{k-1} f(n_i)$. This general result clearly implies Lemma 21.*

From Lemma 21 it follows that the tree constructed by Huffman’s algorithm minimizes any linear function of either $\sum_i n_i$ or $\sum_i \log n_i$, which proves, using Theorem 19, the following corollary.

Corollary 23 *Let \bar{R}_i be the R_i of Equation (28) with every $|\mathcal{S}_i|$ replaced by its maximal value, k^D . Then, $R_{\text{DECO}}(\mathbf{x}_1^n, P_T) \leq \sum_i \bar{R}_i(\mathbf{x}_1^n)$ and the Huffman coding tree minimizes this bound. The resulting bound is given in Corollary 25.*

4. Mind the Gap

Here we compare our redundancy (upper) bound for DECO and the known bound for multi-CTW. Relying on Corollary 23, we focus on the case where DECO uses the Huffman tree.

A clear advantage of the DECO algorithm is that it “activates” only internal node (binary) predictors corresponding to observed symbols. This can be seen by the bound of Theorem 19, which decreases with the number of unobserved symbols. Since the multi-CTW bound is insensitive to alphabet sparsity, this suggests that DECO will outperform the multi-CTW when predicting sequences in which alphabet symbols are sparse.

In this section we prove that the redundancy bound of DECO is strictly better than the corresponding multi-CTW bound, for any sufficiently long sequence. For this purpose, we examine the difference between the two bounds using a worst-case expression of the DECO bound.

Let Σ be an alphabet with $|\Sigma| = k$ and \mathbf{x}_1^n be a sequence over Σ . Fix some order D and let \mathcal{S} be the topology corresponding to the D -bounded tree-source that maximizes

the probability of \mathbf{x}_1^n over \mathcal{C}_D . Denote by \bar{R}_{CTW} the multi-CTW redundancy bound (see Theorem 9),

$$\bar{R}_{\text{CTW}}(\mathbf{x}_1^n) = \frac{(k-1)|\mathcal{S}|}{2} \log \frac{n}{|\mathcal{S}|} + |\mathcal{S}| \log k + \frac{k|\mathcal{S}| - 1}{k-1}. \quad (36)$$

Similarly, let \bar{R}_{HUFF} denote the redundancy of DECO applied with a Huffman-tree (see Theorem 19),

$$\bar{R}_{\text{HUFF}}(\mathbf{x}_1^n) = \sum_{i=1}^{k-1} \left(\frac{\Psi}{2} \log \frac{n_i}{\Psi} + \Psi + \frac{k\Psi - 1}{k-1} \right), \quad (37)$$

where Ψ is an upper-bound on the model-sizes $|\mathcal{S}_i|$ (see Equation 28). We would like to bound below the gap $\bar{R}_{\text{CTW}} - \bar{R}_{\text{HUFF}}$ between these bounds.

The next lemma and corollary provide a worst case upper bound for \bar{R}_{HUFF} .

Lemma 24 *Let \mathbf{x}_1^n be a sequence over Σ . Let T be the corresponding Huffman decomposition tree and $\{n_i\}_{i=1}^{k-1}$ its internal node counters. Then,*

$$\sum_{i=1}^{k-1} \log n_i < (k-1) \cdot (\log n + \log(1 + \log k) - \log(k-1)) \quad (38)$$

Proof Recall that for every symbol $\sigma \in \Sigma$, N_σ denote the number of occurrences of σ in \mathbf{x}_1^n and $\ell(\sigma)$ denotes the length of the path from the root of T to the leaf σ . Denote by \hat{H} the empirical entropy, $\hat{H} = -\sum_{\sigma \in \Sigma} \frac{N_\sigma}{n} \log \frac{N_\sigma}{n}$.

$$\sum_{i=1}^{k-1} \frac{1}{k-1} \log n_i \leq \log \left(\sum_{i=1}^{k-1} \frac{1}{k-1} \log n_i \right) \quad (39)$$

$$= \log \left(\sum_{i=1}^{k-1} \log n_i \right) - \log(k-1)$$

$$= \log \left(\sum_{\sigma \in \Sigma} N_\sigma \ell(\sigma) \right) - \log(k-1) \quad (40)$$

$$< \log \left(n \cdot (1 + \hat{H}) \right) - \log(k-1) \quad (41)$$

$$\leq \log \left(n \cdot (1 + \log k) \right) - \log(k-1) \quad (42)$$

$$= \log n + \log(1 + \log k) - \log(k-1). \quad (43)$$

In (39) we used Jensen's inequality; (40) is an application of Equation (35); T yields a Huffman code with an average code length of $\sum_{\sigma \in \Sigma} \frac{N_\sigma}{n} \ell(\sigma) < 1 + \hat{H}$ (see, e.g., Cover and Thomas, 1991, Section 5.4 and 5.8), which implies (41); finally, (42) follows from the fact that $\hat{H} \leq \log k$ (see, e.g., Cover and Thomas, 1991, Theorem 2.6.4). We conclude by multiplying both sides by $k-1$. \blacksquare

Corollary 25

$$\bar{R}_{\text{HUFF}}(\mathbf{x}_1^n) < \frac{(k-1)\Psi}{2} \left(\log \frac{n}{\Psi} + \log(1 + \log k) - \log(k-1) + 2 + \frac{2k}{k-1} \right).$$

Proof

$$\begin{aligned}
\bar{R}_{\text{HUFF}}(\mathbf{x}_1^n) &= \sum_{i=1}^{k-1} \left(\frac{\Psi}{2} \log \frac{n_i}{\Psi} + \Psi + \frac{k\Psi - 1}{k-1} \right) \\
&= \sum_{i=1}^{k-1} \left(\frac{\Psi}{2} \log n_i \right) + \sum_{i=1}^{k-1} \left(-\frac{\Psi}{2} \log(\Psi) + \Psi + \frac{k\Psi - 1}{k-1} \right) \\
&= \frac{\Psi}{2} \sum_{i=1}^{k-1} (\log n_i) + (k-1) \left(-\frac{\Psi}{2} \log(\Psi) + \Psi + \frac{k\Psi - 1}{k-1} \right) \\
&< \frac{(k-1)\Psi}{2} (\log n + \log(1 + \log k) - \log(k-1)) + \\
&\quad (k-1) \left(-\frac{\Psi}{2} \log(\Psi) + \Psi + \frac{k\Psi - 1}{k-1} \right) \tag{44} \\
&= \frac{(k-1)\Psi}{2} (\log n + \log(1 + \log k) - \log(k-1)) + (k-1) \left(-\frac{\Psi}{2} \log(\Psi) + \Psi + \frac{k\Psi - 1}{k-1} \right) \\
&= \frac{(k-1)\Psi}{2} \left(\log \frac{n}{\Psi} + \log(1 + \log k) - \log(k-1) \right) + (k-1) \left(\Psi + \frac{k\Psi - 1}{k-1} \right) \\
&< \frac{(k-1)\Psi}{2} \left(\log \frac{n}{\Psi} + \log(1 + \log k) - \log(k-1) + 2 + \frac{2k}{k-1} \right). \tag{45}
\end{aligned}$$

Here (44) follows by application of (38) and we obtained (45) using $\frac{k\Psi-1}{k-1} < \frac{k\Psi}{k-1}$. ■

The next theorem characterizes cases where the DECO algorithm has a strictly smaller redundancy bound than the multi-CTW bound.

Theorem 26 *Let Σ be an alphabet with $|\Sigma| = k \geq 118$ and \mathbf{x}_1^n be a sequence over Σ generated by the (unknown) D -bounded multi-source $\mathcal{M} = (\mathcal{S}, P_{\mathcal{S}})$. Then, $\bar{R}_{\text{CTW}}(\mathbf{x}_1^n) > \bar{R}_{\text{HUFF}}(\mathbf{x}_1^n)$.*

Proof We take the upper bound $\Psi = |\mathcal{S}|$. By the proof of Lemma 15, when translating \mathcal{M} into its equivalent decomposed source, the internal node topologies are first initiated with \mathcal{S} and then may be pruned to achieve minimality. Hence, $|\mathcal{S}|$ is an upper bound on the sizes

$|\mathcal{S}_i|$. Thus, we have

$$\begin{aligned}
\bar{R}_{\text{CTW}}(\mathbf{x}_1^n) - \bar{R}_{\text{HUFF}}(\mathbf{x}_1^n) &= \frac{(k-1)|\mathcal{S}|}{2} \log \frac{n}{|\mathcal{S}|} + |\mathcal{S}| \log k + \frac{k|\mathcal{S}| - 1}{k-1} - \\
&\quad \sum_{i=1}^{k-1} \left(\frac{|\mathcal{S}|}{2} \log \frac{n_i}{|\mathcal{S}_i|} + |\mathcal{S}_i| + \frac{k|\mathcal{S}_i| - 1}{k-1} \right) \\
&> \frac{(k-1)|\mathcal{S}|}{2} \log \frac{n}{|\mathcal{S}|} + |\mathcal{S}| \log k + \frac{k|\mathcal{S}| - 1}{k-1} - \frac{(k-1)|\mathcal{S}|}{2} \times \\
&\quad \left(\log \frac{n}{|\mathcal{S}|} + \log(1 + \log k) - \log(k-1) + 2 + \frac{2k}{k-1} \right) \quad (46) \\
&= |\mathcal{S}| \log k + \frac{k|\mathcal{S}| - 1}{k-1} - \\
&\quad \frac{(k-1)|\mathcal{S}|}{2} \left(\log(1 + \log k) - \log(k-1) + 2 + \frac{2k}{k-1} \right), \quad (47)
\end{aligned}$$

where (46) is by Corollary 25. Using straightforward analysis it is not hard to show that (47) grows with k and is positive for $k \geq 118$. This completes the proof. \blacksquare

The gap, between the CTW and DECO bounds, shown in Theorem 26 is relevant when the internal node redundancies of DECO are $R_i = \frac{|\mathcal{S}_i|}{2} \log \frac{n_i}{|\mathcal{S}_i|} + |\mathcal{S}_i| + \frac{k|\mathcal{S}_i| - 1}{k-1}$. By a simple analysis of Equation (28) using the function $f(x) = \frac{x}{2} \log \frac{n}{x} + x + \frac{kx-1}{k-1}$, we can show that the gap is positive when $n_i \geq \max\{0.17 \cdot \Psi, \mathcal{S}_i\}$.

We conclude that the redundancy bound of DECO algorithm converges faster than the bound of the CTW algorithm for alphabet of size $k \geq 118$. Currently, the CTW algorithm is known to have the best convergence rate (see Table 5). Therefore, the current bound is the tightest one known for prediction (and lossless compression) in realistic settings.

Remark 27 *The result of Theorem 26 is obtained using a worst-case analysis for the DECO redundancy. This analysis considered a sequence that contains all alphabet symbols; each symbol appears sufficiently many times. However, in many practical applications (such as predictions of ASCII sequences) most of the symbols are expected to have small frequencies (e.g., by Zipf's Law). In this case, the DECO redundancy is even smaller than the worst case bound of Corollary 25 and the gap between the two bounds is larger.*

5. Examining Other Alphabet Decompositions

The bound \bar{R}_{HUFF} , given in Equation (37), is optimized using a Huffman decomposition tree (Corollary 23). However, replacing each $|\mathcal{S}_i|$ with its maximal value can affect the bound considerably. For example, if we manage to place a very easy (binary) prediction problem at the root, it could be the case that the “true” model order for this problem is very small. Such considerations are not explicitly treated by the Huffman tree optimization. Therefore, it is of major interest to consider other types of alphabet decomposition trees. Also, if our goal is to utilize the (successful) *binary* CTW in multi-alphabet problems, there is no apparent reason why we should restrict ourselves to *hierarchical* alphabet decompositions

as discussed so far. The parallel study of “multi-category decompositions” in supervised learning suggests other approaches such *one-vs-all*, *all-pairs*, etc. (see, e.g., Allwein et al., 2001).

We empirically targeted two questions: (i) Are there better alphabet decomposition trees for the DECO algorithm? (ii) Can the “flat” decomposition techniques of supervised learning be effectively applied in our sequential prediction setting?

To answer the first question, we developed a simple heuristic procedure that attempts to increase log-likelihood performance of the DECO algorithm, starting from any decomposition tree. This procedure searches for a locally optimal tree using the actual performance of DECO on a given sequence. Starting from a given tree, this procedure attempts to swap an alphabet symbol from one subtree to the other while recursively “optimizing” the resulting subtrees. Each such swap is ‘accepted’ only if it improves the actual performance. We applied this procedure using a Huffman tree as the starting point and refer to the resulting algorithm as ‘Improved’.

Sequence	Random	Improved	Huffman	Huffman Comb	Inverted Huffman-Comb
bib	1.91	1.81	1.83	2.04	2.16
news	2.47	2.34	2.36	2.65	2.75
book1	2.26	2.20	2.21	2.28	2.38
book2	1.99	1.92	1.94	2.06	2.14
paper1	2.40	2.26	2.27	2.58	2.69
paper2	2.31	2.21	2.23	2.41	2.53
paper3	2.60	2.45	2.47	2.74	2.87
paper4	2.95	2.72	2.75	3.20	3.34
paper5	3.12	2.86	2.89	3.42	3.56
paper6	2.50	2.32	2.36	2.67	2.84
trans	1.52	1.40	1.43	1.71	1.89
progc	2.51	2.32	2.35	2.76	2.87
progl	1.74	1.64	1.67	1.88	2.01
progp	1.78	1.63	1.66	1.92	2.09
Average	2.29	2.15	2.17	2.45	2.58

Table 1: Comparing average log-loss of DECO with different decomposition structures. The best results appear in boldface. Results for the random decomposition reflect an average on ten random trees.

We experimented with DECO, ‘Improved,’ and several others decomposition schemes. Following standard convention in the lossless compression community, we examined the algorithms over the ‘Calgary Corpus’. This Corpus serves as a standard benchmark for testing log-loss prediction and lossless compression algorithms (Bell et al., 1990; Witten and Bell, 1991; Cleary and Teahan, 1995; Begleiter et al., 2004). The corpus consists of 18 files of nine different types. Most of the files are pure ASCII files and four are binary files. The ASCII files consist of English texts (books 1-2 and papers 1-6), a bibliography

file (bib), a batch of unedited news articles (news), some source code of computer programs (prog c,l,p), and a transcript of a terminal session (trans). The longest file (book1) has 785kb symbols and the shortest (paper5) 12kb symbols.

In addition to the Huffman and ‘Improved’ decompositions, we include the performance of a random tree and two types of “Huffman Comb” trees. The random tree was constructed bottom-up in agglomerative random fashion where symbol cluster pairs to be merged were selected uniformly at random among all available nodes at each ‘merge’ step. Each of the two ‘comb’ trees is a full (binary) tree of height $k - 1$. That is, such trees operate similarly to *decision lists*. The comb tree whose leaves (symbols) are ordered top-down according to their ascending frequencies in \mathbf{x}_1^n is referred to as the “Huffman Comb,” and the comb tree whose leaves are reversely ordered is called the “Inverted Huffman Comb.” Obviously, it is expected at the outset that the inverted Huffman comb will give rise to inferior performance.

In all the experimental results below we analyzed the statistical significance of pairwise comparisons between algorithms using the Wilcoxon signed rank test (Wilcoxon, 1945)⁵ with a confidence level of 95%.

Table 1 shows the average prediction performance of DECO compared to several tree structures over the text files of the Calgary Corpus. The slightly better but statistically significant performance of the improved-DECO indicates that there are more effective trees than Huffman’s. It is also interesting to see that the random tree (based on an average of 10 random trees) is significantly better than both the Huffman Comb trees. The latter observation suggests that it is hard to construct very inefficient decomposition structures.

Sequence*10%	DECO	All-Pairs	One-vs-All
progc	3.11	4.28	4.04
progl	1.66	2.27	2.16
progp	2.69	3.53	3.50
paper1	3.08	3.82	3.67
paper2	3.15	3.66	3.62
paper3	3.39	4.10	4.00
paper4	3.89	4.62	4.54
paper5	3.91	4.82	5.02
paper6	3.32	4.11	4.00
Average	3.13	3.91	3.84

Table 2: Comparing three decomposition methods over a reduced version of the Calgary Corpus. The best results appear in boldface.

To investigate the second question, regarding other decomposition schemes, we implemented the ‘one-vs-all’ and ‘all-pairs’ schemes, straightforwardly adapted to our sequential setting. The reader is referred to Rifkin and Klautau (2004) for a discussion of these techniques in standard supervised learning. The prediction results, over a reduced version of the

5. The Wilcoxon signed rank test is a nonparametric alternative to the paired t-test, which is similar to the Fisher sign test. This test assumes that there is information in the magnitudes of the differences between paired observations, as well as the signs.

Calgary text files, appear in Table 2. In this reduced dataset we took 10% (from the start) of each original sequence. The reason for considering smaller texts (of shorter sequences) is the excessive memory requirements of the ‘all-pairs’ algorithm, which requires $\binom{k}{2} = 8128$ different binary predictors (compared to the $k - 1$ and k binary predictors required by DECO and ‘one-vs-all’, respectively).⁶ The results of Table 2 indicate that the hierarchical decomposition is better than the other two flat decomposition schemes. (Note that the advantage of ‘one-vs-all’ over ‘all-pairs’ is at 90% confidence.)

6. On Applying CTW with Other Zero-Order Estimators

Another interesting direction when attempting to improve the performance of the standard CTW on multi-alphabet sequences is to use other, perhaps stronger (in some sense), zero-order estimators instead of the KT estimator. In particular, it seems most appropriate to consider well-known estimators such as Good-Turing and the very recent ones proposed by OrLitsky et al. (2003), some of which have strong performance guarantees in a certain worst case sense.

To this end, we compared the prediction quality of multi-CTW and DECO each applied with four different sequential zero-order estimators: Good-Turing (denoted \hat{z}^{GT}), “Improved add-one” (denoted \hat{z}^{+1}), “improved Good-Turing” (denoted \hat{z}^{GT^*}) and standard KT (denoted \hat{z}^{KT}). The description of the first three estimators is provided in Appendix B.

All four estimators have worst-case performance guarantees based on a maximal *likelihood ratio*, which is the ratio between the highest possible probability assigned by some distribution and the probability assigned by the estimators. The set of “all possible distributions” considered is referred to as the comparison class. OrLitsky et al. analyzed the performance of these estimators for *infinite* discrete alphabets and a comparison class consisting of *all* possible distributions over n -length sequences. They showed that the average *per-symbol* ratio is infinite for sequential add-constant estimators such as KT. The Good-Turing and Improved add-one estimators assign to each (‘large’) sequence a probability which is at most a factor of c^n (for some constant $c > 1$) smaller than the maximal possible probability; the improved Good-Turing estimator assigns to each sequence a probability that is within a sub-exponential factor of the maximal probability.

In addition to the above, the KT and Good-Turing estimators enjoy the following guarantees. In Theorem 7 we stated a *finite-sample* guarantee for the redundancy of the KT estimator. Recall that this guarantee refers to *finite* alphabets and a comparison class consisting of zero-order distributions. Moreover, within this setting, KT was shown to be (asymptotically) close, up to a constant, to the best possible ratio (Xie and Barron, 2000; Freund, 2003), and the constant is proportional to the alphabet size. Thus, when considering the per-symbol ratio, KT is *asymptotically* optimal. Along with the above worst-case guarantees, the Good-Turing estimator also has a convergence guarantee to the “true” missing mass probability (McAllester and Schapire, 2000), assuming the existence of a true underlying distribution that generated the sequence.

In Tables 3 and 4 we provide the respective per symbol log-loss obtained with these estimators for all the textual (ASCII) sequences from the Calgary Corpus (14 datasets). In

6. With our two gigabyte RAM machine the runs with the entire corpus would take approximately two months.

Sequence	\hat{z}^{KT}	$\hat{z}^{+\perp}$	\hat{z}^{GT}	\hat{z}^{GT^*}
bib	2.47	2.35	2.27	2.29
news	2.92	2.82	2.75	2.75
book1	2.50	2.46	2.42	2.42
book2	2.32	2.24	2.19	2.20
paper1	2.98	2.83	2.73	2.75
paper2	2.77	2.68	2.60	2.61
paper3	3.16	3.08	3.00	2.99
paper4	3.57	3.50	3.41	3.38
paper5	3.76	3.66	3.57	3.56
paper6	3.10	2.95	2.84	2.85
trans	2.18	1.92	1.76	1.84
progc	3.04	2.89	2.79	2.82
progl	2.29	2.14	2.05	2.08
progp	2.26	2.11	2.00	2.04
Average	2.80	2.69	2.60	2.61

Table 3: Comparing the average log-loss of multi-CTW with different sequential zero-order estimators. The comparison is made with textual ($|\Sigma| = 128$) sequences taken from the Calgary Corpus, and with parameter $D = 5$. Each numerical value is the average log-loss (the loss per symbol). The best (minimal) result of each comparison is marked in boldface.

all the experiments below we analyzed the statistical significance of the results using the Wilcoxon signed rank test at a confidence of 95%.

Table 3 presents the log-loss of the four zero-order estimators when used as the zero-order predictor within the multi-CTW scheme. The support set of the zero-order estimators is of size 128. Observe that multi-CTW with \hat{z}^{KT} suffers the worst log-loss. On the other hand, when applying these estimators in DECO (thus, when solving binary prediction problems), as depicted in Table 4, the \hat{z}^{KT} outperforms all the other estimators. Also observe that the best multi-CTW result (\hat{z}^{GT} in Table 3) is worse than the best DECO result (\hat{z}^{KT} in Table 4).

In summarizing these results, we note that:

- For text sequences, the CTW algorithm can be significantly improved when applied with the Good-Turing estimator (instead of the KT estimator).
- The improved Good-Turing estimator proposed by Orłitsky et al. (2003) does not improve the Good-Turing.
- The Deco-Huffman algorithm achieves best performance with the original (binary) KT estimator.

Sequence	\hat{z}^{KT}	\hat{z}^{+1}	\hat{z}^{GT}	\hat{z}^{GT^*}
bib	1.84	2.39	2.02	2.35
news	2.36	2.94	2.54	2.85
book1	2.22	2.39	2.23	2.38
book2	1.94	2.27	2.02	2.26
paper1	2.28	3.03	2.53	2.93
paper2	2.23	2.74	2.39	2.68
paper3	2.47	3.08	2.66	2.98
paper4	2.75	3.52	3.00	3.36
paper5	2.90	3.78	3.18	3.59
paper6	2.36	3.16	2.63	3.04
trans	1.43	2.43	1.83	2.35
progc	2.35	3.16	2.61	3.03
progl	1.67	2.33	1.90	2.26
progp	1.66	2.44	1.95	2.37
Average	2.18	2.83	2.39	2.74

Table 4: Comparing predictions of DECO with different sequential zero-order estimators. The comparison is made with textual ($|\Sigma| = 128$) sequences taken from the Calgary Corpus, and with parameter $D = 5$. Each numerical value resemble the average log-loss (the loss per-symbol). The best (minimal) result of each comparison is marked in boldface.

7. Related Work

To the best of our knowledge, hierarchical alphabet decompositions in the log-loss prediction/compression setting were first considered by Tjalkens, Willems and Shtarkov (1994).⁷ In this paper, the authors study a hierarchical decomposition where each internal node in the decomposition tree is associated with a (binary) KT estimator (instead of binary-CTW instances in DECO). In this setting the comparison class is the set of all zero order sources. The authors derived a redundancy bound of $k - 1 + \frac{1}{2} \sum_{n_i > 0} \log n_i$ for this algorithm, where the n_i terms are the node counters as defined in Theorem 19. This result is similar to a special case of our bound, $2 + \frac{1}{2} \sum_{n_i > 0} \log n_i$, obtained using Theorem 19 for the special case $D = 0$ (implying $|\mathcal{S}_i| = 1$). In that paper Tjalkens et al. proposed the essence of the DECO algorithm as presented here; however, they did not provide the details. A thorough study of algorithm DECO and other CTW-based approaches for dealing with multi-alphabets are presented in Volf’s Ph.D. thesis (Volf, 2002). In particular, an in-depth empirical study of DECO, over the Calgary and Canterbury Corpora, indicated that this algorithm achieves state-of-the-art performance in lossless compression. Thus, it matches the good perfor-

7. A similar paper by Tjalkens, Volf and Willems, proposing the same method and results, appeared a few years later (Tjalkens et al., 1997).

mance of the *prediction by partial match* (PPM) family of heuristics.⁸ Further empirical evidence that substantiated this observation appears in Sadakane et al. (2000); Shkarin (2002); Begleiter et al. (2004).

There are also many discrete prediction algorithms that are not CTW-based. We restrict the discussion here to some of the most popular algorithms that are known to be universal with respect to some comparison class. Probably the most famous (and the first) universal lossless compression algorithms were proposed by Ziv and Lempel (1977; 1978). For example, the well-known LZ78 algorithm is a fast dictionary method that avoids explicit statistical considerations. This algorithm is universal (with respect to the set of ergodic sources); however, in contrast to both conventional wisdom and the algorithm’s phenomenal commercial success, it is not among the best lossless compressors (see, e.g., Bell et al., 1990).

Two more recent universal algorithms are the Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994) and grammar-based compression (Yang and Kieffer, 2000). The public-domain version of BWT, called BZIP, is considered to be a relatively strong compressor over the Calgary Corpus, which is fast but somewhat inferior to PPM and DECO. The grammar-based compression algorithm has the advantage of providing an “explanation” (grammar) for the way it compressed the target sequence.

The point-wise (worst case) redundancy of the prediction game was introduced by Shtarkov (1987). Given a comparison class \mathcal{C} of target distributions P and some hypothesis class \mathcal{P} , from which the prediction algorithm selects one approximating distribution \hat{P} , the point-wise redundancy of this game is

$$R_n^*(\mathcal{C}) = \inf_{\hat{P} \in \mathcal{P}} \sup_{P \in \mathcal{C}} \max_{\mathbf{x}_1^n} \log \frac{P(\mathbf{x}_1^n)}{\hat{P}(\mathbf{x}_1^n)}.$$

Shtarkov also presented the first asymptotic lower bound on the redundancy for the case where both the hypothesis and comparison classes are the set D -order Markov sources. To date, the tightest *asymptotic* lower bound on the point-wise redundancy for D -gram Markov sources was recently given by Jacquet and Szpankowski (2004, Theorem 3). They showed that for large (but unspecified) n , the lower bound is $\frac{|\mathcal{S}|(k-1)}{2} \log \frac{n}{2\pi} + \log A(D, k) + \log(1 + O(\frac{1}{n}))$, where $|\mathcal{S}| = k^D$ and $A(D, k)$ is a constant depending on the order D and the alphabet size k . In Table 5 we present known upper-bounds (leading term) on the redundancy of the algorithms mentioned above. As can be seen, the CTW algorithm enjoys the tightest bound. Note that there exist sequential prediction algorithms that enjoy other types of performance guarantees. One example is the *probabilistic suffix trees* (PST) algorithm (Ron et al., 1996). The PST is a well-known algorithm that is mainly used in the bioinformatic community (see, e.g., Bejerano and Yona, 2001). The algorithm enjoys a PAC-like performance guarantee with respect to the class of VMMs (which is valid only if the predicted sequence was generated by a VMM).

8. As far as we know, the best PPM performance over the Calgary Corpus is reported for the PPM-II variant, proposed by Shkarin (2002).

Algorithm	Per Symbol Point-wise Redundancy	Comparison Class	Source
LZ78	$O(1/\log n)$	Markov Sources	Savari (1997); Kieffer and Yang (1999); Potapov (2004)
CTW	$\frac{ \mathcal{S} (\Sigma -1)}{2n} \log n$	Markov sources	Willems et al. (1995); Willems (1998)
BWT Grammar Based	$\frac{ \mathcal{S} (\Sigma +1)}{2n} \log n$ $O(\log \log n / \log n)$	D -order Markov sources Ergodic sources	Effros et al. (2002) Yang and Kieffer (2000)
Asymptotic Lower Bound	$\frac{ \mathcal{S} (\Sigma -1)}{2n} \log n$	D -order Markov sources	Shtarkov (1987)

Table 5: Point-wise redundancy (leading term) of several universal lossless compression (and prediction) algorithms. The predicted sequence is of length n .

8. Concluding Remarks

Our main result is the first redundancy bound for the DECO algorithm. Our bounding technique can be adapted to DECO-like decomposition schemes using any binary predictor that has a (binary) point-wise redundancy bound with respect to VMMS. To the best of our knowledge, our bound for the Huffman decomposition algorithm (proposed by Volf) is the tightest known for prediction under the log-loss and therefore, for lossless compression. This result provides a compelling justification for the superior empirical performance of the DECO-Huffman predictor/compressor as indicated in several works (see, e.g., Volf, 2002; Sadakane et al., 2000).

Our experiments with random decomposition structures indicate that the DECO scheme is quite robust to the choice of the tree, and even a random tree is likely to outperform the multi-CTW. However, the excellent performance of the Huffman decomposition clearly motivates attempts to optimize it. Our local optimization procedure is able to generate better trees than Huffman’s, suggesting that better prediction can be obtained with better optimization of the tree structure. Similar observations were also reported in Volf (2002). Since finding the best decomposition is an NP-hard problem, a very interesting research question is whether one could optimize the DECO redundancy bound over the possible decompositions.

Interestingly, our numerical examples strongly indicate that hierarchical decompositions are better suited to sequential prediction than the standard ‘flat’ approaches (‘one-vs-all’ and ‘all-pairs’) commonly used in supervised learning. This result may further motivate the consideration of hierarchical decompositions in supervised learning (e.g., as suggested by Huo et al., 2002; Cheong et al., 2004; El-Yaniv and Etzion-Rosenberg, 2004).

The fact that the other zero-order estimators can improve the multi-CTW performance (with larger alphabets) motivates further research along these lines. First, it would be interesting to try combining CTW with other zero-order estimators. Second, it would be interesting to analyze the combined algorithm(s), possibly by relying on the worst case results of Orlitsky et al. (2003).

But perhaps the most important research target at this time is the development of a lower bound on the redundancy of predictors for finite (and short) sequences. While the Jacquet and Szpankowski (2004) lower bound is indicative on the asymptotical achievable rates, it is meaningless in the finite (and small) sample context. For example, our bounds, and even the multi-CTW bounds known today, are smaller than the Jacquet and Szpankowski *lower* bound.

9. Acknowledgments

We thank Paul A. Volf and Roe Engelberg for the helpful discussions and Tjalling J. Tjalkens for providing relevant bibliography.

References

- E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2001. ISSN 1533-7928.
- R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.
- G. Bejerano and G. Yona. Variations on probabilistic suffix trees: Statistical modeling and the prediction of protein families. *Bioinformatics*, 17(1):23–43, 2001.
- T.C. Bell, J.G. Cleary, and I.H. Witten. *Text Compression*. Prentice-Hall, Inc., 1990.
- M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
- O. Catoni. Statistical learning theory and stochastic optimization. *Lecture Notes in Mathematics*, 1851, 2004.
- S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318, 1996.
- S. Cheong, S.H. Oh, and S. Lee. Support vector machines with binary tree architecture for multi-class classification. *Neural Information Processing - Letters and Reviews*, 2(3): 47–51, March 2004.
- J.G. Cleary and W.J. Teahan. Experiments on the zero frequency problem. In *DCC '95: Proceedings of the Conference on Data Compression*, page 480, Washington, DC, USA, 1995. IEEE Computer Society.
- R. Courant and F. John. *Introduction to Calculus and Analysis*. Springer-Verlag, 1989.
- T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

- M. Effros, K. Visweswariah, S.R. Kulkarni, and S. Verdu. Universal lossless source coding with the Burrows Wheeler transform. *IEEE Transactions on Information Theory*, 48(5):1061–1081, 2002.
- R. El-Yaniv and N. Etzion-Rosenberg. Hierarchical multiclass decompositions with application to authorship determination. Technical Report CS-2004-15, Technion - Israel Institute of Technology, March 2004.
- Y. Freund. Predicting a binary sequence almost as well as the optimal biased coin. *Information and Computation*, 182(2):73–94, 2003.
- C.R. Glassey and R.M. Karp. On the optimality of Huffman trees. *SIAM Journal on Applied Mathematics*, 31(2):368–378, September 1976.
- I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 1953.
- D. Haussler, J. Kivinen, and M.K. Warmuth. Sequential prediction of individual sequences under general loss functions. *IEEE Transactions on Information Theory*, 44(5):1906–1925, 1998.
- D.P. Helmbold and R.E. Schapire. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning*, 27(1):51–68, 1997.
- A. Hodges. *Alan Turing: the enigma*. Walker and Co., 2000.
- X. Huo, J. Chen, S. Wang, and K.L. Tsui. Support vector trees: Simultaneously realizing the principles of maximal margin and maximal purity. Technical report, The Logistics Institute, Georgia Tech, and Asia Pacific, National University of Singapore, 2002.
- P. Jacquet and W. Szpankowski. Markov types and minimax redundancy for Markov sources. *IEEE Transactions on Information Theory*, 50:1393–1402, 2004.
- J.C. Kieffer and E.H. Yang. A simple technique for bounding the pointwise redundancy of the 1978 Lempel-Ziv algorithm. In *DCC '99: Proceedings of the Conference on Data Compression*, page 434. IEEE Computer Society, 1999.
- R. Krichevsky and V. Trofimov. The performance of universal encoding. *IEEE Transactions on Information Theory*, 27:199–207, 1981.
- P. Laplace. Philosophical essays on probabilities. *Springer-Verlag*, 1995. Translated by A. Dale from the 5th (1825) edition.
- D. McAllester and R. Schapire. On the convergence rate of Good-Turing estimators. In *In Proceedings of the Thirteenth annual conference on computational learning theory*, 2000.
- N. Merhav and M. Feder. Universal prediction. *IEEE Transactions on Information Theory*, 44(6):2124–2147, 1998.
- A. Orlitsky, N.P. Santhanam, and J. Zhang. Always Good Turing: Asymptotically optimal probability estimation. *Science*, 302(5644):427–431, October 2003.

- C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- V.N. Potapov. Redundancy estimates for the Lempel-Ziv algorithm of data compression. *Discrete Applied Mathematics*, 135(1-3):245–254, 2004. ISSN 0166-218X.
- R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.
- D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2-3):117–149, 1996.
- K. Sadakane, T. Okazaki, and H. Imai. Implementing the context tree weighting method for text compression. In *Data Compression Conference*, pages 123–132, 2000.
- S.A. Savari. Redundancy of the Lempel-Ziv incremental parsing rule. *IEEE Transactions on Information Theory*, 43:9–21, 1997.
- D. Shkarin. PPM: One step to practicality. In *Data Compression Conference*, pages 202–212, 2002.
- Y. Shtarkov. Universal sequential coding of single messages. *Problems in Information Transmission*, 23:175–186, 1987.
- N.J.A. Sloane and S. Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.
- T.J. Tjalkens, Y. Shtarkov, and F.M.J. Willems. Context tree weighting: Multi-alphabet sources. In *Proc. 14th Symp. on Info. Theory, Benelux*, pages 128–135, 1993.
- T.J. Tjalkens, P.A. Volf, and F.M.J. Willems. A context-tree weighting method for text generating sources. In *Data Compression Conference*, page 472, 1997.
- T.J. Tjalkens, F.M.J. Willems, and Y. Shtarkov. Multi-alphabet universal coding using a binary decomposition context tree weighting algorithm. In *Proc. 15th Symp. on Info. Theory, Benelux*, pages 259–265, 1994.
- P.A. Volf. *Weighting Techniques in Data Compression Theory and Algorithms*. PhD thesis, Technische Universiteit Eindhoven, 2002.
- V. Vovk. Aggregating strategies. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, pages 371–383, 1990.
- F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.
- F.M.J. Willems. The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, 44(2):792–798, March 1998.
- F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, pages 653–664, 1995.
- I.H. Witten and T.C. Bell. The zero-frequency problem: estimating the probabilities of novelevents in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094, 1991.

Q. Xie and A.R. Barron. Asymptotic minimax regret for data compression, gambling, and prediction. *IEEE Transactions on Information Theory*, 46(2):431–445, 2000.

E.H. Yang and J.C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform. part one: Without context models. *IEEE Transactions on Information Theory*, 46(3):755–777, 2000.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, May 1977.

J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.

Appendix A. On the KT Estimator - Proof of Theorem 7

We provide a proof for the (worst-case) performance guarantee of the KT estimator as stated in Theorem 7. This proof is based on lecture notes by Catoni (2004)⁹

Lemma 28 Consider the case where KT counts all the symbols of the sequence \mathbf{x}_1^n (i.e., $s = \epsilon$). Then,

$$\hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n) = \frac{\Gamma(\frac{k}{2}) \prod_{\sigma \in \Sigma} \Gamma(N_\sigma + \frac{1}{2})}{\Gamma(\frac{1}{2})^k \Gamma(\sum_{\sigma \in \Sigma} N_\sigma + \frac{k}{2})}, \quad (48)$$

where $\Gamma(x) = \int_{\mathbb{R}^+} t^{x-1} \exp(-t) dt$ is the gamma function.¹⁰

Proof

The proof is based on the identity $\Gamma(x+1) = x\Gamma(x)$ and on a rewriting of Definition 6,

$$\begin{aligned} \hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n) &= \hat{\mathbf{z}}^{\text{KT}}(x_1^{n-1}) \frac{N_{x_n} + 1/2}{\sum_{\sigma \in \Sigma} N_\sigma + k/2} \\ &= \frac{\prod_{\sigma \in \Sigma} ((1/2) \cdot (1 + 1/2) \cdot (2 + 1/2) \cdots (N_\sigma + 1/2))}{(k/2) \cdot (1 + k/2) \cdot (2 + k/2) \cdots (\sum_{\sigma \in \Sigma} N_\sigma + k/2)} \\ &= \frac{\prod_{\sigma \in \Sigma} \left(\frac{1}{\Gamma(\frac{1}{2})^k} \Gamma(N_\sigma + \frac{1}{2}) \right)}{\frac{1}{\Gamma(\frac{k}{2})} \Gamma(\sum_{\sigma \in \Sigma} N_\sigma + \frac{k}{2})}, \end{aligned}$$

and (48) is obtained by rearranging the terms. ■

We now provide a proof for Theorem 7. Recall that this theorem states an upper bound of $\frac{k-1}{2} \log n + \log k$ for the worst-case redundancy of $\hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n)$.

9. Krichevsky and Trofimov (1981) proved an asymptotic version of Theorem 7 for the average redundancy; Willems et al. (1995) provided a proof for binary alphabets.

10. It can be shown that $\Gamma(1) = 1$ and that $\Gamma(x+1) = x\Gamma(x)$. Therefore, if $n \geq 1$ is an integer, $\Gamma(n+1) = n!$. For further information see, e.g., (Courant and John, 1989).

Proof It is sufficient to prove that

$$\frac{\hat{\mathbf{z}}^{\text{KT}}(\mathbf{x}_1^n)}{\sup_{\mathbf{z} \in \mathcal{Z}} \mathbf{z}(\mathbf{x}_1^n)} n^{\frac{k-1}{2}} \geq \frac{1}{k}. \quad (49)$$

Let $\mathbf{a} = (a_i)_{i=1}^k \in \mathbb{N}^k$ be a vector of arbitrary symbol counts for some sequence \mathbf{x}_1^n .¹¹ For these counts, by Lemma 28, KT would assign the probability, $\frac{\Gamma(\frac{k}{2}) \prod_{i=1}^k \Gamma(a_i + \frac{1}{2})}{\Gamma(\frac{1}{2})^k \Gamma(\sum_{i=1}^k a_i + \frac{k}{2})}$. Let \mathbf{z} be the corresponding empirical distribution: $\mathbf{z}(\mathbf{x}_1^n) = \prod_{i=1}^k \left(\frac{a_i}{\sum_{i=1}^k a_i} \right)^{a_i}$, where $n = \sum_{i=1}^k a_i$. It is well known that, given the counts \mathbf{a} , the distribution that maximizes the probability of \mathbf{x}_1^n is \mathbf{z} , the maximum likelihood distribution (see, e.g., Cover and Thomas, 1991, Theorem 12.1.2). Thus, taking $\mathbf{z} = \arg \max_{\mathbf{z}' \in \mathcal{Z}} \mathbf{z}'(\mathbf{x}_1^n)$, inequality (49) becomes

$$\Delta(\mathbf{a}) = \frac{\Gamma(\frac{k}{2}) \prod_{i=1}^k \Gamma(a_i + \frac{1}{2}) (\sum_{i=1}^k a_i)^{\sum_i a_i + \frac{k-1}{2}}}{\Gamma(\frac{1}{2})^k \Gamma(\sum_{i=1}^k a_i + \frac{k}{2}) \prod_{i=1}^k a_i^{a_i}} \geq \frac{1}{k}.$$

We have to show that for any $\mathbf{a} \neq (0, 0, \dots, 0)$, $\Delta(\mathbf{a}) \geq \frac{1}{k}$.

Observe that $\Delta(\mathbf{a})$ is invariant under any permutation of the coordinations of \mathbf{a} . Also note that, by the identity, $\Gamma(x+1) = x\Gamma(x)$,

$$\Delta((1, 0, \dots, 0)) = \frac{\Gamma(\frac{k}{2})\Gamma(1 + \frac{1}{2})}{\Gamma(\frac{1}{2})\Gamma(1 + \frac{k}{2})} = \frac{1}{k}.$$

It is sufficient to prove that for any $\mathbf{a} = (a_1, a_2, \dots, a_n)$ with $a_1 > 1$, $\Delta(\mathbf{a}) \geq \Delta((a_1 - 1, a_2, \dots, a_k))$. Observe that

$$\Delta(\mathbf{a}) = \Delta(a_1 - 1, a_2, \dots, a_k) \frac{(a_1 - \frac{1}{2})(a_1 - 1)^{a_1 - 1}}{a_1^{a_1}} \frac{n^{n + \frac{k-1}{2}}}{(n + \frac{k}{2} - 1)(n - 1)^{n-1 + \frac{k-1}{2}}}.$$

Thus, it is enough to show that

$$\frac{(a_1 - \frac{1}{2})(a_1 - 1)^{a_1 - 1}}{a_1^{a_1}} \frac{n^{n + \frac{k-1}{2}}}{(n + \frac{k}{2} - 1)(n - 1)^{n-1 + \frac{k-1}{2}}} \geq 1, \quad \text{where } a_1 \geq 1, n \geq 2.$$

This can be done by showing that

$$\begin{aligned} f(t) &= \log \left(\frac{(t - \frac{1}{2})(t - 1)^{t-1}}{t^t} \right) \geq -1; \\ g(q) &= \log \left(\frac{q^{q + \frac{k-1}{2}}}{(q + \frac{k}{2} - 1)(q - 1)^{q-1 + \frac{k-1}{2}}} \right) \geq 1. \end{aligned}$$

Recall that $\lim_{x \rightarrow +\infty} (1 + \frac{y}{x})^x = e^y$ and observe that,

$$\begin{aligned} \lim_{t \rightarrow +\infty} f(t) &= \lim_{t \rightarrow +\infty} \log \left(1 - \frac{1}{2t} \right) + (t - 1) \log \left(1 - \frac{1}{t} \right) = -1; \\ \lim_{q \rightarrow +\infty} g(q) &= \lim_{q \rightarrow +\infty} - \left(q - 1 + \frac{k-1}{2} \right) \log \left(1 - \frac{1}{q} \right) - \log \left(1 + \frac{k-2}{2q} \right) = 1. \end{aligned}$$

11. In information theory \mathbf{a} is called a *type*. See, e.g., (Cover and Thomas, 1991, Chapter 12).

We conclude by showing that both functions decreasing monotone to their limits, hence, $f'(t) \leq 0$ and $g'(q) \leq 0$. For $f'(t)$ we next show that it is a non-decreasing function (i.e., $f''(t) \geq 0$) that is bounded from above by zero.

$$\begin{aligned} f'(t) &= \frac{1}{t - \frac{1}{2}} + \log\left(\frac{t-1}{t}\right); \\ \lim_{t \rightarrow +\infty} f'(t) &= 0; \\ f''(t) &= \frac{-1}{(t - \frac{1}{2})^2} + \frac{1}{t(t-1)}; \\ &= \frac{-1}{(t - \frac{1}{2})^2} + \frac{1}{(t - \frac{1}{2})^2 - \frac{1}{4}} \geq 0. \end{aligned}$$

Therefore, $f'(t) \geq 0$ for any $t > 1$. In a similar manner,

$$\begin{aligned} g'(q) &= \log\left(\frac{q}{1-q}\right) + \frac{k-1}{2} \left(\frac{1}{q} - \frac{1}{q-1}\right) - \frac{1}{q + \frac{k}{2} - 1}; \\ \lim_{q \rightarrow +\infty} g'(q) &= 0; \\ g''(q) &\geq 0. \end{aligned}$$

■

Appendix B. Zero Order Estimators

In this appendix we describe the sequential zero-order estimators: Good-Turing, “Improved add-one”, “improved Good-Turing” by their “next-symbol” probability assignment. These estimators are compared along with KT in Section 6.

Recall that, by the chain-rule, $\hat{\mathbf{z}}(\mathbf{x}_1^n) = \prod_{t=0}^{n-1} \hat{z}(x_{t+1}|\mathbf{x}_1^t)$, where \mathbf{x}_1^0 is the empty sequence. Hence, it is sufficient to define the next-symbol prediction, $\hat{z}(x_{t+1}|\mathbf{x}_1^t)$, which is based on the symbol counts N_σ in \mathbf{x}_1^t . We require the following definition. Let \mathbf{x}_1^t be a sequence. Define a_m to be the number of symbols that appear exactly m times in \mathbf{x}_1^t , i.e., $a_m = |\{\sigma \in \Sigma : N_\sigma = m\}|$. We denote the “improved add-one” estimator by $\hat{\mathbf{z}}^{+1}$ and the “improved GT” by $\hat{\mathbf{z}}^{\text{GT}*}$.¹²

The Good-Turing (GT) estimator (Good, 1953) is well known and most commonly used in language modeling for speech recognition (see, e.g., Chen and Goodman, 1996).¹³ The next-symbol probability generated by GT is

$$\hat{z}^{\text{GT}}(\sigma|\mathbf{x}_1^t) = \frac{1}{S_{t+1}} \times \begin{cases} \frac{a'_1}{t \times a_0}, & \text{if } N_\sigma = 0; \\ \frac{N_\sigma + 1}{t} \frac{a'_{N_\sigma + 1}}{a'_{N_\sigma}}, & \text{otherwise,} \end{cases} \quad (50)$$

12. In Orlitsky et al. (2003) the $\hat{\mathbf{z}}^{+1}$ estimator is denoted by q_{+1} and $\hat{\mathbf{z}}^{\text{GT}*}$ by $q_{1/3}$.

13. I.J. Good and A.M. Turing used this estimator to break the Enigma Cipher (Hodges, 2000) during World War II.

where a'_m is a smoothed version of a_m and S_{t+1} is a normalization factor.¹⁴ In the following experiments we used the simple smoothing suggested by Orlitsky et al. (2003) where $a'_m = \max(a_m, 1)$.

Denote by m the number of distinct symbols in \mathbf{x}_1^t (i.e., $m = \sum_{i=1}^t a_i$). The next-symbol probability of the improved add-one estimator is

$$\hat{z}^{+1}(\sigma|\mathbf{x}_1^t) = \frac{1}{S_{t+1}} \times \begin{cases} \frac{m+1}{a_0}, & \text{if } N_\sigma = 0; \\ (t - m + 1) \frac{N_\sigma}{t}, & N_\sigma > 0, \end{cases} \quad (51)$$

where S_{t+1} is a normalization factor.

For any natural number c , define the function $f_c(a) = \max(a, c)$. Also define the integer-sequence $c_n = \lceil n^{1/3} \rceil$. The next-symbol probability assigned by the improved GT estimator is

$$\hat{z}^{\text{GT}^*}(\sigma|\mathbf{x}_1^t) = \frac{1}{S_{t+1}} \times \begin{cases} \frac{f_{c_{t+1}}(a_1+1)}{a_0}, & \text{if } N_\sigma = 0; \\ (N_\sigma + 1) \frac{f_{c_{t+1}}(a_{N_\sigma+1})}{f_{c_{t+1}}(a_{N_\sigma})}, & \text{otherwise,} \end{cases} \quad (52)$$

where S_{t+1} is a normalization factor. The improved GT estimator (\hat{z}^{GT^*}) is optimal with respect to the worst-case criterion of Orlitsky et al. (2003).

14. Orlitsky et al. mention that Turing had an intuitive motivation for this estimator. Unfortunately, this explanation was never published.