

# Optimal Parsing Trees for Run-Length Coding of Biased Data

Sharon Aviran, Paul H. Siegel, and Jack K. Wolf  
University of California, San Diego  
La Jolla, CA 92093, USA  
Emails: {saviran, psiegel, jwolf}@ucsd.edu

**Abstract**—We study coding schemes which encode unconstrained sequences into run-length-limited  $(d, k)$ -constrained sequences. We present a general framework for the construction of such  $(d, k)$ -codes from variable-length source codes. This framework is an extension of the previously suggested bit stuffing, bit flipping and symbol sliding algorithms. We show that it gives rise to new code constructions which achieve improved performance over the three aforementioned algorithms. Therefore, we are interested in finding optimal codes under this framework, optimal in the sense of maximal achievable asymptotic rates. However, this appears to be a difficult problem. In an attempt to solve it, we are led to consider the encoding of unconstrained sequences of independent but biased (as opposed to equiprobable) bits. Here, our main result is that one can use the Tunstall source coding algorithm to generate optimal codes for a partial class of  $(d, k)$  constraints.

## I. INTRODUCTION

A binary sequence satisfies a *run-length-limited*  $(d, k)$  constraint if it has the following two properties: successive ones are separated by at least  $d$  zeros and the number of consecutive zeros does not exceed  $k$ . Such sequences are called  $(d, k)$ -sequences and have found widespread use in magnetic and optical recording [1]. In constrained-code design, we typically model the unconstrained user-data as a stream of independent equiprobable bits. The focus of this paper is on the design of efficient codes which convert such inputs into  $(d, k)$ -sequences. Here, efficiency relates to asymptotic (asym.) encoding rates, where the rate  $R$  of a  $(d, k)$ -code is defined as the ratio of the average input length and the average output length. The rate is evaluated with respect to the *Shannon capacity* of the  $(d, k)$  constraint, a measure which was shown by Shannon to equal the maximum rate achievable by any  $(d, k)$ -code [2]. It is an established result [2] that for any admissible  $(d, k)$  pair, the capacity exists and is given by  $C(d, k) = \log_2 \lambda_{d,k}$ , where  $\lambda_{d,k}$  is the largest real root of the constraint's characteristic polynomial. The most efficient code is thus a code whose rate equals the capacity. We say that such a code *achieves capacity* or is *capacity-achieving*.

The central theme of this work is twofold: a study of prior  $(d, k)$ -code constructions from a source coding perspective and the construction of new  $(d, k)$ -codes based on variable input-length (VL) source codes. The idea of constructing  $(d, k)$ -codes from source codes is not new. For example, by reversing a source encoder-decoder pair, the decoder of a suitable source code can be used to encode unconstrained

Bernoulli(1/2)-distributed bitstreams into  $(d, k)$ -sequences in a recoverable manner [3] [4]. In such designs, the choice of source code is guided by special properties of  $(d, k)$ -sequences with maximum-entropy (*maxentropic*) distribution. Such sequences are desirable as they correspond to maximizing the code rate [2]. It is well-known that they can be parsed into a concatenation of binary strings from the set  $\Gamma_{d,k} = \{0^d 1, 0^{d+1} 1, \dots, 0^{k-1} 1, 0^k 1\}$ , where the strings are statistically independent and identically distributed (i.i.d.) [2]. From now on, we refer to the strings in  $\Gamma_{d,k}$  as *constrained phrases*. The constrained-phrase maxentropic distribution is given by  $\Lambda_{d,k} = (\lambda_{d,k}^{-(d+1)}, \lambda_{d,k}^{-(d+2)}, \dots, \lambda_{d,k}^{-(k)}, \lambda_{d,k}^{-(k+1)})$ , where  $\lambda_{d,k}^{-(d+i+1)}$  is the probability of  $0^{d+i} 1$ . The source code then serves as a *distribution transformer* (DT) between  $\Lambda_{d,k}$  and a Bernoulli(1/2) distribution. The  $(d, k)$ -code thus applies the inverse transformation so as to induce  $\Lambda_{d,k}$  on the output.

An alternative approach emerges from the literature on lossless coding for transmission over noiseless, memoryless channels with unequal symbol-transmission costs. One can accommodate  $(d, k)$ -codes into this framework by modelling  $(d, k)$ -sequences as the outputs of a special memoryless channel [1]. Existing literature is mainly concerned with two types of source codes: fixed-to-variable length and variable-to-fixed length (VFL), the latter being sparsely studied. Our work relates to the second type. In this case, Lempel, Even, and Cohn [5] derived an algorithm for constructing a prefix-free code of minimum average transmission cost per source symbol when the source symbols are equiprobable. Here, we relax the equiprobable source assumption.

This work builds upon three prior  $(d, k)$ -code constructions: the *bit stuffing* (BS), *bit flipping* (BF), and *symbol sliding* (SS) algorithms. In all constructions, the input sequences are first fed into a *binary* DT. The DT preserves the bitwise independence, but results in bits that are *biased* towards one of two values, i.e., they constitute an i.i.d. Bernoulli( $p$ )-distributed source. As these sequences are still unconstrained, they undergo additional processing by a *constrained encoder*. It is this component which varies among the algorithms. Note that although one can directly  $(d, k)$ -encode the standard equiprobable input, it turns out that the introduction of a bias into the data is key to achieving improved rates [6]. Intuitively speaking, it better conforms the data to the characteristics of maxentropic sequences. It is also worth noting that the

binary DT is a special case of general DT's, such as the ones introduced in [3] [4]. In fact, some of these source coding techniques can be readily applied to the binary case, where a Bernoulli( $p$ ) distribution replaces  $\Lambda_{d,k}$ . Hence, a direct transformation to  $\Lambda_{d,k}$  requires a similar implementation to a binary DT. Still, the schemes presented here provide alternative methods of approximating  $\Lambda_{d,k}$ . The challenge here is to approximate it with a non-conventional source, while using simple techniques.

The rest of the paper is organized as follows. In Section II, we review the BS, BF and SS algorithms as well as the Tunstall algorithm [7], which generates VFL source codes of minimal compression ratio. Section III is devoted to studying  $(d, k)$ -codes that are based on VL source codes. We first examine the BS, BF, and SS algorithms within the context of a general framework for constructing  $(d, k)$ -codes from source codes (Sec. III-A). We demonstrate that the framework gives rise to new improved code constructions. This prompts us to search for optimal codes, optimal in the sense of maximal achievable asym. rates (Sec. III-B). Nevertheless, finding such codes is a difficult problem. We therefore resort to studying a simpler related problem, where we seek an optimal  $(d, k)$ -code for a biased source (Sec. III-C). Interesting properties of optimal  $(d, k)$ -codes arise, leading to a solution to the simpler problem for a subclass of  $(d, k)$  constraints, based upon the Tunstall algorithm. Related open problems are also discussed.

## II. BACKGROUND AND RELATED WORK

### A. The bit stuffing, bit flipping and symbol sliding algorithms

The *bit stuffing algorithm* (BS) [6] sequentially writes an unconstrained bitstream while inserting extra bits whenever the constraint might be violated. The encoder consists of a *binary distribution transformer* (DT) followed by a *bit stuffer*. The DT bijectively converts a sequence of i.i.d. *unbiased* ( $Pr\{0\} = \frac{1}{2}$ ) bits into a *p-biased* sequence of independent Bernoulli bits, whose probability of a 0 is some  $p \in [0, 1]$ ,  $p \neq \frac{1}{2}$ . Throughout, we refer to  $p$  as the *bias*. The asym. expected rate of such conversion is  $h(p)$ , where  $h(p)$  is the binary entropy function. The  $p$ -biased sequence is then fed into the bit stuffer, which tracks the number of consecutive zeros in the *encoded* sequence. Once it equals  $k$ , a 1 followed by  $d$  0's are inserted (*stuffed*). Whenever encountering a biased 1,  $d$  0's are stuffed. The decoder applies similar logic to identify and discard the stuffed bits. The inverse DT then recovers the unbiased input from the  $p$ -biased sequence.

The expected rate of the BS scheme is the product of the expected rates of the two components. A tradeoff between these two rates requires one to optimize  $p$  in order to maximize the average overall rate. In [6], Bender and Wolf showed that by judiciously biasing at the first step, the algorithm achieves capacity when  $k = d + 1$  or  $k = \infty$ , and is near-capacity achieving for many other  $(d, k)$  pairs. A more recent work [8] modified BS by adding a controlled flipping of unconstrained bits before writing them. The modification, named the *bit flipping algorithm* (BF), was shown to achieve

improved average rates over BS for most  $(d, k)$  constraints. Additionally, it achieves capacity only for the  $(2, 4)$  constraint.

In a subsequent work, Sankarasubramanian and McLaughlin [9] generalized both BS and BF into an improved construction, called the *symbol sliding algorithm* (SS). Their key insight was an interpretation of BS and BF as repeatedly applying certain bijective mappings between the strings in the set  $T_{BS}^{d,k} = \{1, 01, \dots, 0^{k-d-1}1, 0^{k-d}\}$  and the phrases in  $\Gamma_{d,k}$ . SS is essentially an adjustment of the mapping to obtain further improved rates. Specifically, *symbol sliding with index  $j$*  (SS( $j$ )) corresponds to the mapping indicated by the arrows between the two leftmost columns in Fig. 1. The fourth column lists the probabilities that are induced on the statistically independent constrained phrases. It can be shown [9] that BS and BF are special cases of symbol sliding with indices  $j = 0$  and  $j = 1$ , respectively. The idea behind modifying the mapping is that it changes the distribution that is induced on the constrained phrases, and may provide a better match to their maxentropic distribution. For example, the probabilities induced by BS appear in the third column. Starting with the BS-induced distribution, SS( $j$ ) amounts to sliding  $p^{(k-d)}$  up by  $j$  positions, while pushing each of  $p^{(k-d-j)}(1-p), \dots, p^{(k-d-1)}(1-p)$  down by one position (see Fig. 1). The sliding index  $j$  serves as an additional parameter which needs to be optimized, but in turn provides more flexibility in fitting the distribution to  $\Lambda_{d,k}$ . Indeed, SS demonstrates rate gains over BF for several constraints and additionally achieves capacity for all  $(d, 2d + 1)$  constraints.

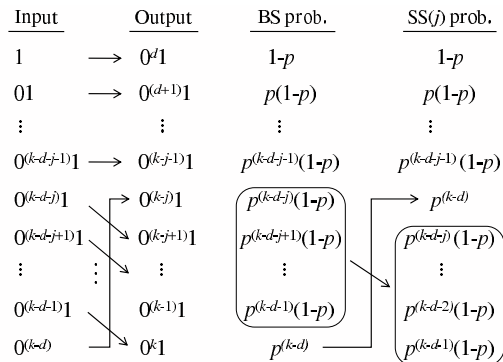


Fig. 1. Mapping and induced probabilities of symbol sliding with index  $j$ .

### B. The Tunstall algorithm

A variable-to-fixed length (VFL) code partitions an  $M$ -ary source sequence into a concatenation of variable-length  $M$ -ary *source words* that are encoded into *uniform-length codewords*. The code is defined by specifying a source-word set  $T = \{w_1, \dots, w_K\}$  and a bijective assignment of the codewords to the source words. Throughout this paper, we restrict attention to codes that use exhaustive and prefix-free source-word sets. We shall work with their tree representations, which are called *parsing trees*. A typical source coding problem is to find a VFL code that minimizes the compression ratio, defined as the ratio of average output and average input lengths. We say

that such a code is *VFL-optimal*. Additionally, we assume a memoryless and stationary source  $S = \{s_1, \dots, s_M\}$ , which is ruled by a given probability distribution  $P = \{p_1, \dots, p_M\}$ . Under this assumption, the problem reduces to maximizing the expected input length  $L_T^{in}(P) = \sum_{w_i \in T} Pr(w_i, P) \cdot L(w_i)$ , where  $L(w)$  stands for the length of a string  $w$  and  $Pr(w, P)$  represents its probability [7].

In [7], Tunstall provided a simple procedure to construct a VFL-optimal parsing tree for any valid parsing tree size  $|T| = K$ . The idea is to grow the tree from the top down by successively extending it from the leaf of largest probability. The algorithm goes as follows.

- 1) Initialize: let  $T := S$  be the tree containing the root and its  $M$  children. The leaves of  $T$  correspond to the  $M$  source letters and their probabilities are listed in  $P$ .
- 2) While  $|T| \neq K$ , repeat the following operations on  $T$ :
  - Select a leaf  $w_i \in T$  with maximal probability and add its  $M$  children to the tree,
  - Compute the leaf probabilities for the extended tree.

### III. A SOURCE CODING PERSPECTIVE

#### A. Binary-transformer algorithms revisited

Consider a system that encodes  $p$ -biased sequences into  $(d, k)$ -sequences. The encoder parses the input stream into binary *source words*, and subsequently replaces each source word with a *constrained phrase* from  $\Gamma_{d,k}$ . The  $(d, k)$ -code, thus, has two parameters: a set of source words  $T = \{w_1, \dots, w_{k-d+1}\}$  and a bijective assignment  $f : T \rightarrow \Gamma_{d,k}$ . In our model, the input's zero-memory and stationarity extends to the source-word sequences as well as to the constrained-phrase sequences. One can easily compute the probability distribution  $P_T(p) = \{Pr(w_1, p), \dots, Pr(w_{k-d+1}, p)\}$  that is induced on  $T$  as well as on  $\Gamma_{d,k}$ . The *asymptotic average information rate of the  $(d, k)$ -code* is

$$R_{T,f}^{d,k}(p) = \frac{L_T^{in}(p)}{L_{T,f}^{out}(p)} = \frac{\sum_{w_i \in T} Pr(w_i, p) \cdot L(w_i)}{\sum_{w_i \in T} Pr(w_i, p) \cdot L(f(w_i))}. \quad (1)$$

The problem of interest is finding a code that maximizes the rate for a given  $p$ . Such a code is said to be  $(d, k)$ -*optimal*. Although the problem is mentioned in the literature [5], to the best of our knowledge, it has not been treated for general  $p$ 's.

Similarly, we can derive  $(d, k)$ -codes for systems that operate on unbiased data and utilize a DT to introduce a bias into the data before parsing takes place. The problem then is to find a pair  $(p, (T, f))$  of a bias and a parsing-tree code that jointly maximize the *asymptotic average overall rate* of the system, given by  $I_{T,f}^{d,k}(p) = R_{T,f}^{d,k}(p) \cdot h(p)$ . Before we proceed to examine parsing trees in more detail, we point out the following useful property of such  $(d, k)$ -optimal codes.

*Lemma 1:* Given a bias  $p$  and a  $(d, k)$ -code  $(T, f)$ , let  $v_i = f^{-1}(0^{d+i-1}1)$  for all  $1 \leq i \leq k - d + 1$ . If  $V = (v_1, \dots, v_{k-d+1})$  satisfies the condition

$$Pr(v_1, p) \geq Pr(v_2, p) \geq \dots \geq Pr(v_{k-d+1}, p) \quad (2)$$

then  $R_{T,f}^{d,k}(p)$  is maximum over all assignments of  $\Gamma_{d,k}$  to  $T$ .

Lemma 1 implies that a search for a maximal-rate code need only account for the one code that optimizes the assignment for each of the candidate parsing trees. Such a code assigns the shortest phrase to the most probable word, the next shortest phrase to the second most probable word, and so on. We can thus omit the assignment  $f$  when referring to a code.

Recall that in Section II-A, we interpreted BS, BF and SS as particular mappings between the strings in  $T_{BS}^{d,k} = \{1, 01, \dots, 0^{k-d-1}1, 0^{k-d}\}$  and the phrases in  $\Gamma_{d,k}$ . Observing that  $T_{BS}^{d,k}$  is exhaustive and prefix-free, we can view BS, BF and SS as special cases of codes under the above-described framework. The sole difference between the algorithms is their specified assignments of the constrained phrases to the input words, leading to different constrained-phrase probabilities. From now on, we shall refer to  $T_{BS}^{d,k}$  as the *BS tree*.

Consider now an optimal assignment for the BS tree. It is attained by listing the input words in order of non-increasing probability and by assigning increasingly longer phrases to them. However, we have seen that the ordering varies with  $p$ , and consequently, so does the optimal assignment. A search for this assignment is implicitly performed by the SS algorithm. The sliding of  $p^{(k-d)}$  up to any index  $j > 0$  (see Fig. 1) attempts to rearrange the induced probabilities in the desired order. It can be shown that when optimizing for  $j$ , the algorithm slides  $p^{(k-d)}$  up to its proper position in the ordered set. In summary, BS and BF apply fixed assignments irrespectively of the bias. In contrast, the extension to SS results in optimized assignment per given bias, and can thus potentially achieve improved rates over the former two algorithms.

Having optimized both the assignment and the bias per the BS tree, we wish to examine the achievable rates associated with other parsing trees. For a given bias, each parsing tree of size  $k - d + 1$  may be considered in conjunction with its optimal assignment. We now remind the reader that the SS algorithm was motivated by the idea that a judicious shuffling of the BS probabilities could result in an improved match to the maxentropic vector  $\Lambda_{d,k}$  [9]. Nevertheless, can other trees induce probabilities that provide an even better match? In what follows, we are initially interested in finding the tree and bias that jointly maximize the overall rate of a scheme which includes a DT. We then address a somewhat simpler problem of finding the  $(d, k)$ -optimal tree when the bias  $p$  is given. Although the first problem is more interesting in the context of bit stuffing, an efficient solution to the second problem may simplify the solution and analysis of the first.

#### B. Jointly optimal bias and parsing-tree code

Although one can obtain a closed-form expression for the average rate associated with each tree and each bias, the complexity of the rate expressions makes analysis intractable. For that reason, numerical optimization was carried out, considering all possible parsing trees of size  $k - d + 1$  and all biases  $p$  such that  $0 < p < 1$ . In addition, the fast-growing number of candidate trees confined the search to small values of  $k - d$ . For each such value, a broad range of  $(d, k)$  pairs was considered. Table I shows optimal trees for numerous  $(d, k)$

TABLE I  
OPTIMAL PARSING-TREE CODES FOR VARIOUS  $(d, k)$  CONSTRAINTS

$(d, d+3)$ Constraint	Best Tree	$(d, d+4)$ Constraint	Best Tree	$(d, d+5)$ Constraint	Best Tree
(0,3)	1	(0,4)	BS	(0,5)	BS
(1,4)	SS	(1,5)	BF	(1,6)	5
(2,5)	SS	(2,6)	3	(2,7)	6
(3,6)	SS	(3,7)	SS	(3,8)	SS
(4,7)	SS	(4,8)	3	(4,9)	SS
		(5,9)	3	(5,10)	SS
				(6,11)	7
$5 \leq d \leq 30$	2	$6 \leq d \leq 30$	4	$7 \leq d \leq 30$	8

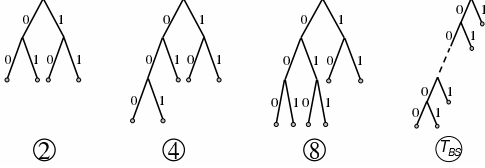


Fig. 2. The asymptotically optimal trees for  $(d, d+3)$ ,  $(d, d+4)$  and  $(d, d+5)$  constraints, and the general form of the BS tree.

constraints, where  $k-d=3, 4$  and  $5$ . We refer to the trees by arbitrary labels, where Fig. 2 depicts trees number 2, 4, 8 as well as the general form of the BS tree and Fig. 3 depicts all other optimal trees. Additionally, for the BS tree, we point out the most efficient of the three algorithms.

It can be seen from Table I that for many constraints, there exists a code construction which outperforms SS. This means that certain trees give rise to constrained-phrase distributions which provide a better match to the maxentropic vector  $\Lambda_{d,k}$  than the BS-tree's induced distribution. For example, when  $k-d=3$  and  $5 \leq d \leq 30$ , the distribution  $P_2(p) = \{(1-p)^2, p(1-p), p(1-p), p^2\}$  leads to improved performance over the BS distribution  $P_{BS}^{d,d+3}(p) = \{1-p, p(1-p), p^2(1-p), p^3\}$ .

Another interesting effect is a convergence towards a specific  $(d, k)$ -optimal tree, starting from a certain  $d$ . Although we have not proved that this indeed holds for  $d$ 's larger than 30, we shall call these trees the *asymptotically optimal trees*. We outline the difference between these trees and the BS tree using the following definitions. We say that a binary tree is a *skew tree* if it is obtained by either consistently extending its rightmost leaf or by consistently extending its leftmost leaf. In contrast, a *balanced tree* of size  $K$  is a binary tree where each subtree of the root is of the same height if  $K=2^D$  for some  $D$ , or where the two subtrees differ in height by at most 1 and are balanced as well if  $K \neq 2^D$ . Clearly, the BS tree is

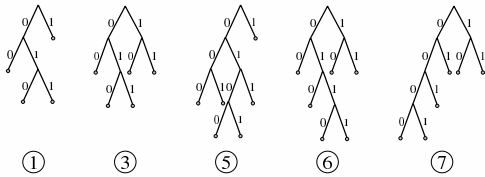


Fig. 3. Other optimal trees for  $(d, d+3)$ ,  $(d, d+4)$  and  $(d, d+5)$  constraints.

a skew tree, while the asym. optimal trees are all balanced.

A few questions arise. Do the asym. optimal trees maintain their optimality as  $d$  approaches infinity? If so, is there an asym. optimal tree for any given  $k-d$ ? Can we characterize these trees, for example, by their skewness? Lastly, how can one efficiently find these trees? These questions are difficult to address without a good insight into the joint optimization problem. In the following subsection we try to pursue a better understanding of the problem by studying a related problem.

### C. Optimal parsing tree codes for a given bias

We next tackle the problem of finding a  $(d, k)$ -optimal tree when the bias is given; that is, we remove the DT from the system. However, this problem is still hard and has not been addressed in prior literature. The only case that was solved algorithmically is when  $p=0.5$  [5]. Therefore, we resorted to an exhaustive search over all possible parsing trees of size  $k-d+1$  for each of the  $(d, k)$  pairs considered before and for each bias  $p \in (0, 1)$ .

Suppose we fix  $k-d$  and we inspect the various rate functions, while gradually increasing  $d$ , starting from  $d=0$ . When examining the  $(d, k)$ -optimal code per bias, we noticed that a fixed pattern emerges once a certain  $d$  value is crossed. Specifically, past this point (and up to  $d=30$ ), it appears that the  $(d, k)$ -optimal tree per bias is fixed, and that the range of biases ( $0 < p < 1$ ) is divided into continuous subintervals, each corresponding to a certain optimal tree. We found that the  $d$  thresholds for  $k-d=3, 4$  and  $5$  are 3, 6 and 6, respectively. Yet, our most interesting finding is that the fixed  $(d, k)$ -optimal tree for each considered bias is, in fact, the Tunstall tree for that bias. We can intuitively explain it as follows. A code maps the input words into phrases of various lengths ranging from  $d+1$  to  $k+1$ . When  $d$  is considerably larger than the fixed  $k-d$ , the variation in phrase lengths is negligible, and they are approximately equal. That said, it only seems reasonable that the VFL-optimal coding scheme will prove to be an efficient scheme in those cases as well. Still, there seems to be more to these observations than the given interpretation, as the observed  $d$  thresholds are comparable to the  $(k-d)$ 's.

It is interesting to find whether the revealed behavior indeed applies to arbitrarily large  $d$ 's and  $(k-d)$ 's. The next lemma settles this question by asserting the conjectured properties.

*Lemma 2:* Let  $m > 0$  be an integer and  $T_{Tun}(p)$  be a Tunstall tree of size  $m+1$  that corresponds to a  $p$ -biased binary memoryless source. Then, there exists an integer  $d_m$ , such that for any  $(d, d+m)$  constraint with  $d \geq d_m$ , the following holds for all  $0 < p < 1$ :

$$R_{T_{Tun}(p)}^{d,d+m}(p) \geq R_T^{d,d+m}(p) \quad (3)$$

for any parsing-tree code  $T$  where  $|T|=m+1$ .

*Proof:* For any  $(d, d+m)$ -code  $T$ , we can write

$$R_T^{d,d+m}(p) = \frac{L_T^{in}(p)}{d + \sum_{i=1}^{m+1} Pr(v_i, p) \cdot i} = \frac{L_T^{in}(p)}{d + L_T^{sub}(p)} \quad (4)$$

where  $v_i = f^{-1}(0^{d+i-1}1)$ , and where both  $L_T^{sub}(p)$  and  $L_T^{in}(p)$  are independent of  $d$ . Now,  $T_{Tun}(p)$  attains the max-

imum rate for any  $p$  if and only if (3) holds for any  $T$  of size  $m + 1$ . Substituting (4) into (3) and rearranging terms, we obtain the following equivalent condition for any  $T$  and  $p$ :

$$d \cdot \left( \frac{1}{L_T^{in}(p)} - \frac{1}{L_{T_{Tun}(p)}^{in}(p)} \right) \geq \frac{L_{T_{Tun}(p)}^{sub}(p)}{L_{T_{Tun}(p)}^{in}(p)} - \frac{L_T^{sub}(p)}{L_T^{in}(p)}. \quad (5)$$

Clearly, the right-hand side of (5) as well as the expression in parenthesis on the left-hand side are independent of  $d$ . Since a Tunstall tree maximizes  $L_T^{in}(p)$ , we have

$$\frac{1}{L_T^{in}(p)} - \frac{1}{L_{T_{Tun}(p)}^{in}(p)} \geq 0 \quad \forall 0 < p < 1. \quad (6)$$

Furthermore, it can be shown that the inequality (6) is strict whenever  $T$  is not a Tunstall tree. Hence, for a large enough  $d$ , the left-hand side of (5) will be greater than its right-hand side for all  $p$ . In case  $T$  is another Tunstall tree (a Tunstall tree is not always unique), one can show that it achieves the same rates as  $T_{Tun}$ . We complete the proof by setting  $d_m$  to the smallest  $d$  for which (5) holds for all parsing trees.  $\square$

In light of Lemma 2, we proceed to examine additional characteristics of Tunstall trees. Here we present results by Fabris *et al.* [10], pertaining to the relationship between the bias and the structure of the Tunstall tree. They define a *Tunstall region* to be the set of all source probability distributions that are optimally encoded by the same Tunstall code. An analysis of the binary case results in a full characterization of these regions. Representing the source distribution by its bias  $p$ , it is proved that the Tunstall regions have the form of continuous subintervals of the unit interval. As noted earlier, there exist biases for which the Tunstall tree is not unique, thus implying that the subintervals are not necessarily disjoint. It can be shown, however, that the performances of the multiple Tunstall codes are the same, hence one can choose a single representative tree per bias. To resolve this ambiguity, Fabris *et al.* slightly modify the Tunstall algorithm, so that it generates a unique tree. Consequently, each  $p$  belongs only to one Tunstall region, and so the regions form a partition of  $(0, 1)$  into distinct subintervals. Fig. 4 shows an example from [10] demonstrating the segmentation of the interval  $(0.5, 1)$  into three Tunstall regions in the case when the tree size is 6. The paper provides a simple method to compute the region boundaries and at the same time, to construct all trees. It is interesting to observe the structure of the two Tunstall trees at the extremes of the half-unit interval. These are always the balanced tree, which is optimal at least for  $p = 0.5$ , and the skew tree (the BS tree), which is optimal in the neighborhood of  $p = 1$ .

The above-mentioned properties of Tunstall codes are especially appealing in the context of the complex optimization problem with which we dealt in Section III-B. For every  $(d, d+m)$  pair where  $d \geq d_m$ , the following two-stage approach greatly simplifies it. First, one can carry out the algorithm described in [10] to compute the Tunstall regions and trees. Subsequently, one can evaluate the rate associated with the proper tree at each bias and proceed to optimize the *overall* rate. This way, optimization is restricted to a limited number

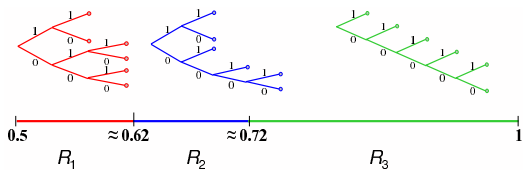


Fig. 4. Tunstall regions with their corresponding trees for  $K = 6$ . After [10].

of Tunstall trees, which can be easily constructed. An upper bound derived in [10] implies that the number of candidate Tunstall trees will not exceed the order of  $(k-d) \log(k-d)$  – a significantly smaller number than the number of all parsing trees. Lemma 2, in conjunction with the results of [10], also provides some insight into the asym. convergence pattern we observed in Table I. The lemma suggests that from a certain  $d$  onwards, only a few fixed Tunstall trees, among which is the BS tree, are competing for the maximum. Moreover, one can verify that the asym. optimal trees in Table I are always the balanced Tunstall trees. Although we can not infer that this will always be the case, we can narrow down the “asym. candidates” to the relatively small set of Tunstall trees.

We have shown that there exists a threshold behavior of the  $(d, k)$  constraints for which the  $(d, k)$ -optimal code is always a Tunstall code. However, we did not compute the  $d$  threshold, nor provide a method for doing so. Moreover, devising a general algorithm that generates the  $(d, k)$ -optimal tree given arbitrary  $d, k$  and  $p$  remains an open problem.

#### ACKNOWLEDGMENT

This research was supported in part by NSF Grant CCR-0219582, part of the Information Technology Research Program and by the CMRR at UCSD.

#### REFERENCES

- [1] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, “Codes for digital recorders,” *IEEE Trans. Inform. Theory*, vol. 44, pp. 2260–2299, Oct. 1998.
- [2] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pt. 1, pp. 379–423, 1948.
- [3] K. J. Kerpez, “Runlength codes from source codes,” *IEEE Trans. Inform. Theory*, vol. 37, no. 3, pp. 682–687, May 1991.
- [4] G. N. N. Martin, G. G. Langdon, and S. J. P. Todd, “Arithmetic codes for constrained channels,” *IBM J. Res. Develop.*, vol. 27, no. 2, pp. 94–106, March 1983.
- [5] A. Lempel, S. Even, and M. Cohn, “An algorithm for optimal prefix parsing of a noiseless and memoryless channel,” *IEEE Trans. Inform. Theory*, vol. 19, no. 2, pp. 208–214, Mar. 1973.
- [6] P. E. Bender and J. K. Wolf, “A universal algorithm for generating optimal and nearly optimal run-length-limited, charge constrained binary sequences,” in *Proc. 1993 IEEE Int. Symp. Inform. Theory*, San Antonio, TX, Jan. 1993, p. 6.
- [7] B. P. Tunstall, “Synthesis of noiseless compression codes,” Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, 1967.
- [8] S. Aviran, P. H. Siegel, and J. K. Wolf, “An improvement to the bit stuffing algorithm,” *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2885–2891, Aug. 2005.
- [9] Y. Sanakarasubramaniam and S. W. McLaughlin, “Capacity achieving code constructions for two classes of  $(d, k)$  constraints,” *submitted to IEEE Trans. Inform. Theory*, June 2004.
- [10] F. Fabris, A. Sgarro, and R. Pauletti, “Tunstall adaptive coding and miscoding,” *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 2167–2180, Nov. 1996.