More Efficient Algorithms and Analyses for Unequal Letter Cost Prefix-Free Coding

Mordecai Golin, *Member, IEEE*, and Jian Li

***Abstract*—There is a large literature devoted to the problem of finding an optimal (min-cost) prefix-free code with an unequal letter-cost encoding alphabet of size. While there is no known polynomial time algorithm for solving it optimally, there are many good heuristics that all provide additive errors to optimal. The additive error in these algorithms usually depends linearly upon the largest encoding letter size.**

This paper was motivated by the problem of finding optimal codes when the encoding alphabet is infinite. Because the largest letter cost is infinite, the previous analyses could give infinite error bounds. We provide a new algorithm that works with infinite encoding alphabets. When restricted to the finite alphabet case, our algorithm often provides better error bounds than the best previous ones known.

***Index Terms*—Entropy, prefix-free codes, redundancy, source-coding.**

I. INTRODUCTION

LET $\Sigma = \{\sigma_1, \sigma_2 \ldots, \sigma_t\}$ be an *encoding alphabet*; Σ^* represents all finite words written using Σ. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a nonempty word. A *code* over Σ is a collection of words $C = \{w_1, \ldots, w_n\}$. Code C is *prefix-free* if for all $i \neq j$ w_i is not a prefix of w_j. See Fig. 1.

Let $cost(w)$ be the *length* or number of characters in w. Given a set of associated probabilities $p_1, p_2, \ldots, p_n \geq 0$, $\sum_i p_i = 1$, the cost of the code is $Cost(C) = \sum_{i=1}^n cost(w_i)p_i$. The *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost. This problem is very well studied and has a well-known $O(tn \log n)$-time greedy algorithm due to Huffman [1] ($O(tn)$-time if the p_i are sorted in non-decreasing order).

Alphabetic coding is the same problem with the additional constraint that the codewords must be chosen in increasing alphabetic order (with respect to the words to be encoded). This corresponds, for example, to the problem of constructing optimal (with respect to average search time) search trees for items

Manuscript received May 1, 2007; revised September 17, 2007. The work of M. Golin was supported in part by HK RGC Competitive Research Grants 613105 and 613507. The work of J. Li was supported in part by the National Natural Science Fund China under Grant 60573025 and was performed while he was visiting the Hong Kong University of Science and Technology on leave from the Shanghai Key Laboratory of Intelligent Information Processing, Department of Computer Science and Engineering, Fudan University.

M. Golin is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: golin@cs.ust.hk).

J. Li is with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA (e-mail: lijian@cs.umd.edu).

Communicated by W. Szpankowski, Associate Editor for Source Coding.

Color versions of Figures 7 and 9 in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2008.926326

x	aaa	aab	ab	b
$cost(x)$	3	5	4	3

x	aaa	aab	ab	$aaba$
$cost(x)$	3	5	4	6

Fig. 1. In this example, $\Sigma = \{a, b\}$. The code on the left is $\{aaa, aab, ab, b\}$ which is prefix-free. The code on the right is $\{aaa, aab, ab, aaba\}$ which is not prefix-free because aab is a prefix of $aaba$. The second row of the tables contain the costs of the codewords when $cost(a) = 1$ and $cost(b) = 3$.

with the given access probabilities or frequencies. Such a code can be constructed in $O(tn^3)$ time [2].

One well studied generalization of the problem is to let the encoding letters have different costs. That is, let $\sigma_i \in \Sigma$ have associated cost c_i. The cost of codeword $w = \sigma_{i_1}\sigma_{i_2}, \ldots, \sigma_{i_l}$ will be $cost(w) = \sum_{k=1}^l c_{i_k}$, i.e., the sum of the costs of its letters (rather than the length of the codeword) with the cost of the code still being defined as $Cost(C) = \sum_{i=1}^n cost(w_i)p_i$ with this new cost function.

The existing, large, literature on the problem of finding a minimal-cost prefix-free code when the c_i are no longer equal, which will be surveyed below, assumes that Σ is a finite alphabet, i.e., that $t = |\Sigma| < \infty$. The original motivation of this paper was to address the problem when Σ is *unbounded*. As will briefly be described in Section II, this models certain types of language restrictions on prefix-free codes and the imposition of different cost metrics on search trees. The tools developed, though, turn out to provide improved approximation bounds for many of the finite cases as well.

More specifically, it was known [3], [4][1] that $\frac{1}{c}H(p_1, \ldots, p_n) \leq OPT$ where

$$H(p_1, \ldots, p_n) = -\sum_{i=1}^n p_i \log p_i$$

is the *entropy* of the distribution, c is the unique positive root of the *characteristic equation* $1 = \sum_{i=1}^t 2^{-cc_i}$ and OPT is the minimum cost of any prefix-free code for those p_i. Note that in this paper, $\log x$ will always denote $\log_2 x$. For $t < \infty$, the known efficient algorithms create a code T that satisfies

$$C(T) \leq \frac{1}{c}H(p_1, \ldots, p_n) + f(\mathcal{C}) \tag{1}$$

where $C(T)$ is the cost of code T, $\mathcal{C} = (c_1, c_2, \ldots, c_t)$, and $f(\mathcal{C})$ is some function of the letter costs $\mathcal{C}$, with the actual value of $f(\mathcal{C})$ depending upon the particular algorithm. Since $\frac{1}{c}H(p_1, \ldots, p_n) \leq OPT$, code T has an *additive error* at most $f(\mathcal{C})$ from OPT. The $f(\mathcal{C})$ corresponding to the different algorithms shared an almost linear dependence upon the value

[1]Note that if $t = 2$ with $c_1 = c_2 = 1$ then $c = 1$ and this reduces to the standard entropy lower bound for prefix-free coding. Although the general lower bound is usually only explicitly derived for finite t, Krause [3] showed how to extend it to infinite t in cases where a positive root of $1 = \sum_{i=1}^\infty 2^{-cc_i}$ exists.

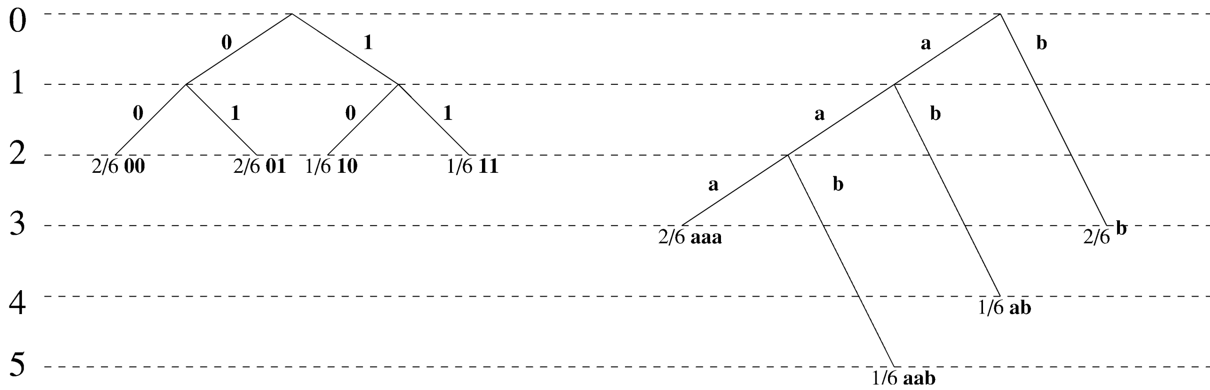

Fig. 2. Two min-cost prefix-free codes for probabilities $2/6, 2/6, 1/6, 1/6$, and their tree representations. The code on the left is optimal for $c_1 = c_2 = 1$ while the code on the right, the prefix-free code from Fig. 1, is optimal for $c_1 = 1, c_2 = 3$.

$c_t = \max(\mathcal{C})$, the largest letter cost. They therefore cannot be used for infinite $\mathcal{C}$. In this paper, we present a new algorithmic variation (most algorithms for this problem start with the same splitting procedure so they are all, in some sense, variations of each other) with a new analysis.

- (Theorems 2 and 3) For finite $\mathcal{C}$, we derive new additive error bounds $f(\mathcal{C})$ which, in many cases, are much better than the old ones.
- (Lemma 9) If $\mathcal{C}$ is infinite but $d_j = |\{m \,|\, j \le c_m < j+1\}|$ is bounded, then we can still give a bound of type (1). For example, if $c_m = 1 + \lfloor \frac{m-1}{2} \rfloor$, i.e., exactly two letters each of length $i, = 1, 2, 3, \ldots$, then we can show that $f(\mathcal{C}) \le 1 + \frac{3}{\log 3}$.
- (Theorem 4) If $\mathcal{C}$ is infinite but d_i is unbounded then we cannot provide a bound of type (1) but, as long as $\sum_{i=1}^{\infty} c_m 2^{-cc_m} < \infty$, we can show that

$$\forall \epsilon > 0, \;\; C(T) \le (1+\epsilon)\frac{1}{c} H(p_1, \ldots, p_n) + f(\mathcal{C}, \epsilon) \quad (2)$$

where $f(\mathcal{C}, \epsilon)$ is some constant based only on $\mathcal{C}$ and ϵ.

We now provide some more history and motivation.

For a simple example, refer to Fig. 2. Both codes written there have minimum cost for the frequencies $(p_1, p_2, p_3, p_4) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6})$ but under different letter costs. The code $\{00, 01, 10, 11\}$ has minimum cost for the standard Huffman problem in which $\Sigma = \{0, 1\}$ and $c_1 = c_2 = 1$, i.e., the cost of a word is the number of bits it contains. The code $\{aaa, aab, ab, b\}$ has minimum cost for the alphabet $\Sigma = \{a, b\}$ in which the cost of an "a" is 1 and the cost of a "b" is 3, i.e., $\mathcal{C} = (1, 3)$.

The unequal letter cost coding problem was originally motivated by coding problems in which different characters have different transmission times or storage costs [5]–[9]. One example is the telegraph channel [3], [10], [11] in which $\Sigma = \{\cdot, -\}$ and $c_1 = 1, c_2 = 2$, i.e., in which dashes are twice as long as dots. Another is the (a, b) run-length-limited codes used in magnetic and optical storage [12], [13], in which the codewords are binary and constrained so that each $\mathbf{1}$ must be preceded by at least a, and at most b, $\mathbf{0}$'s. (This example can be modeled by the unequal-cost letter problem by using an encoding alphabet of $r = b - a + 1$ characters $\{0^k 1 : k = a, a+1, \ldots, b\}$ with associated costs $\{c_i = a + i - 1\}$.)

The unequal letter cost *alphabetic* coding problem arises in designing testing procedures in which the time required by a test depends upon the outcome of the test [14, Sec. 6.2.2, Example 33] and has also been studied under the names *dichotomous search* [15] or the *leaky shower* problem [16].

The literature contains many algorithms for the unequal-cost coding problem. Blachman [5], Marcus [6], and (much later) Gilbert [11] give heuristic constructions without analyses of the costs of the codes they produced. Karp gave the first algorithm yielding an exact solution (assuming the letter costs are integers); Karp's algorithm transforms the problem into an integer program and does not run in polynomial time [7]. Later, exact algorithms based on dynamic programming were given by Golin and Rote [13] for arbitrary t and a slightly more efficient one by Bradford *et al.* [17] for $t = 2$. These algorithms run in $n^{c_t + O(1)}$ time where c_t is the cost of the largest letter. Despite the extensive literature, there is no known polynomial-time algorithm for the generalized problem, nor is the problem known to be NP-hard. Golin, Kenyon, and Young [18] provide a polynomial time approximation scheme (PTAS). Their algorithm is mainly theoretical and not useful in practice. Finally, in contrast to the nonalphabetic case, alphabetic coding has a polynomial-time algorithm $O(tn^3)$ time algorithm [2].

Karp's result was followed by many efficient algorithms [3], [4], [19]–[21]. As mentioned above, $\frac{1}{c}H(p_1, \ldots, p_n) \le OPT$; almost[2] all of these algorithms produce codes of cost at most $C(T) \le \frac{1}{c}H(p_1, \ldots, p_n) + f(\mathcal{C})$ and therefore give solutions within an *additive error* of optimal. An important observation is that the additive error in these papers $f(\mathcal{C})$ somehow incorporate the cost of the largest letter $c_t = \max(\mathcal{C})$. Typical in this regard is Mehlhorn's algorithm [4] which provides a bound of

$$cC(T) - H(p_1, \ldots, p_n) \le (1 - p_1 - p_n) + cc_t. \quad (3)$$

Thus, none of the algorithms described can be used to address infinite alphabets with unbounded letter costs.

The algorithms all work by starting with the probabilities in some given order, grouping consecutive probabilities together

[2]As mentioned by Mehlhorn [4], the result of Cot [20] is a bit different. It is a redundancy bound and not clear how to efficiently implement as an algorithm. Also, the redundancy bound is in a very different form involving taking the ratio of roots of multiple equations that makes it difficult to compare to the others in the literature.

according to some rule, assigning the same initial codeword prefix to all of the probabilities in the same group and then recursing. They therefore actually create alphabetic codes. Another unstated assumption in those papers (related to their definition of alphabetic coding) is that the order of the c_m is given and must be maintained.

In this paper, we are only interested in the general coding problem and not the alphabetic one and will therefore have freedom to dictate the original order in which the p_i are given and the ordering of the c_m. We will actually always assume that $p_1 \ge p_2 \ge p_2 \ge \cdots$ and $c_1 \le c_2 \le c_3 \le \cdots$. These assumptions are the starting point that will permit us to derive better bounds. Furthermore, for simplicity, we will always assume that $c_1 = 1$. If not, we can always force this by uniformly scaling all of the c_i.

For further references on Huffman coding with unequal letter costs, see Abrahams' survey on source coding [22, Sec. 2.7], which contains a section on the problem.

II. Examples of Unequal-Cost Letters

It is very easy to understand the unequal-cost letter problem as modeling situations in which different characters have different transmission times or storage costs [5]–[9]. Such cases will all have finite alphabets. It is not *a priori* as clear why infinite alphabets would be interesting. We now discuss some motivation.

In what follows, we will need some basic language notation. A language $\mathcal{L}$ is just a set of words over alphabet Σ. The *concatenation* of languages A and B is $AB = \{ab \mid a \in A, b \in B\}$. The i-fold concatenation, $\mathcal{L}^i$, is defined by $\mathcal{L}^0 = \{\lambda\}$ (the language containing just the empty string), $\mathcal{L}^1 = \mathcal{L}$, and $\mathcal{L}^i = \mathcal{L}\mathcal{L}^{i-1}$. The *Kleene star* of $\mathcal{L}$ is $\mathcal{L}^* = \bigcup_{i=0}^{\infty} \mathcal{L}^i$.

Example 1: $\mathcal{C} = \{1, 2, 3, \dots, \}$ i.e., $\forall m > 0, c_m = m$.

This is one of the simplest infinite cost vectors. An early use was in [23]. The idea there was to construct a tree (not a code) in which the internal pointers to children were stored in a linked list. Taking the mth pointer corresponds to using character σ_m. The time that it takes to *find* the mth pointer is proportional to the location of the pointer in the list. Thus (after normalizing time units) $c_m = m$.

Example 2: $\mathbf{1}$-ended codes.

The problem of finding min-cost prefix-free codes with the additional restriction that all codewords end with a $\mathbf{1}$ was studied in [24], [25] with the motivation of designing self-synchronizing codes. One can model this problem as follows. Let $\mathcal{L}$ be a language. In our problem,

$$\mathcal{L} = \{w \in \{0,1\}^* \mid \text{the last letter in } w \text{ is a } \mathbf{1}\}. \quad (4)$$

We say that a code C is in $\mathcal{L}$ if $C \subseteq \mathcal{L}$. The problem is to find a minimum cost code among all codes in $\mathcal{L}$.

Note that $\mathcal{L} = \mathcal{Q}^*$ where $\mathcal{Q} = \{1, 01, 001, 0001, \dots\}$. Because $\mathcal{Q}$ is prefix-free (even though $\mathcal{L}$ isnot), every word in $\mathcal{L}$ can be uniquely decomposed as the concatenation of words in $\mathcal{Q}$. If the decomposition of $w \in \mathcal{L}$ is $w = q_1 q_2 \dots q_r$ for $q_i \in \mathcal{Q}$, then $cost(w) = \sum_{i=1}^{r} cost(q_i)$. We can therefore model the problem of finding a minimum cost code among all codes in $\mathcal{L}$ by first creating an infinite alphabet $\Sigma_{\mathcal{Q}} = \{\sigma_q \mid q \in \mathcal{Q}\}$ with associated cost vector $C_{\mathcal{Q}}$ (in which the cost of σ_q is $cost(q)$) and then solving the minimal cost coding problem for $\Sigma_{\mathcal{Q}}$ with those associated costs. For $\mathbf{1}$-ended codes we set $\mathcal{Q}$ as above and thus $\mathcal{C} = \{1, 2, 3, \dots, \}$, i.e., another infinite alphabet with $c_m = m$ for all $m \ge 1$.

Example 3: Σ'-ended unequal letter-cost codes.

The only place above in which we used the specific definition (4) of $\mathcal{L}$ was in the choice of the appropriate $\mathcal{Q}$. In fact, the derivation works for the problem of finding a min-cost prefix-free code in $\mathcal{L}$ where $\mathcal{L}$ is *any* (generally non-prefix-free) language which can be decomposed as $\mathcal{L} = \mathcal{Q}^*$ for some prefix-free language $\mathcal{Q}$.

As an example, consider the following generalization of $\mathbf{1}$-ended codes. Suppose we are given an unequal cost coding problem with *finite* alphabet $\Sigma = \{\sigma_1, \dots, \sigma_t\}$ and associated cost vector $\mathcal{C} = (c_1, \dots, c_t)$. Now let $\Sigma' \subset \Sigma$ and define

$$\mathcal{L} = \Sigma^* \Sigma' = \{w \in \Sigma^* \mid \text{the last letter in } w \text{ is in } \Sigma'\}.$$

$\mathcal{L} = \mathcal{Q}^*$ where $\mathcal{Q} = (\Sigma - \Sigma')^* \Sigma'$ is a prefix-free language We can therefore model the problem of finding a minimum-cost code among all codes in $\mathcal{L}$ by solving an unequal cost coding problem with alphabet $\Sigma_{\mathcal{Q}} = \{\sigma_q \mid q \in \mathcal{Q}\}$ and associated cost vector $C_{\mathcal{Q}}$ (in which $cost(\sigma_q) = cost(q)$). The important observation is that

$$d_j = |\{w \in \Sigma_{\mathcal{Q}} \mid cost(w) = j\}|,$$

the number of letters in $\Sigma_{\mathcal{Q}}$ of cost j, satisfies a linear recurrence relation. Bounding redundancies for these types of $\mathcal{C}$ will be discussed in Section VI, Case 4.

As a specific illustration, consider $\Sigma = \{1, 2, 3\}$ with $\mathcal{C} = (1, 1, 2)$ and $\Sigma' = \{1\}$; our problem is to find minimal cost prefix-free codes in which all words end with a 1. $\mathcal{L} = \{1, 2, 3\}^*\{1\} = \mathcal{Q}^*$, where $\mathcal{Q} = \{2, 3\}^*\{1\}$. The number of characters in $\Sigma_{\mathcal{Q}}$ with cost j is

$$d_1 = 1, \quad d_2 = 1, \quad d_3 = 2, \quad d_4 = 3, \quad d_5 = 5$$

and, in general, $d_{i+2} = d_{i+1} + d_i$, so $d_i = F_i$, the Fibonacci numbers. This specific case will be examined in Section VI, Example 5.

Example 4: Balanced binary words.

We conclude with a very natural $\mathcal{L}$ for which we do *not* know how to analyze the redundancy. In Section VI, Case 5, we will discuss why this is difficult.

Let $\mathcal{L}$ be the set of all "balanced" binary words,[3] i.e., all words which contain exactly as many $\mathbf{0}$'s as $\mathbf{1}$'s. Note that $\mathcal{L} = \mathcal{Q}^*$, where $\mathcal{Q}$ is the set of all nonempty balanced words w such that no prefix of w is balanced. Note that, by definition, $\mathcal{Q}$ is prefix-free. Let

$$\ell_j = |\{w \in \mathcal{L} \mid cost(w) = j\}|$$
$$d_j = |\{w \in \mathcal{Q} \mid cost(w) = j\}|$$

and set $L(z) = \sum_{n=0}^{\infty} \ell_n z^n$ and $D(z) = \sum_{n=0}^{\infty} d_n z^n$ to be their associated generating functions. If $\mathcal{L} = D^*$, then standard

[3]This also generalizes a problem from [26] which provides heuristics for constructing a min-cost prefix-free code in which the expected number of $\mathbf{0}$'s equals the expected number of $\mathbf{1}$'s.

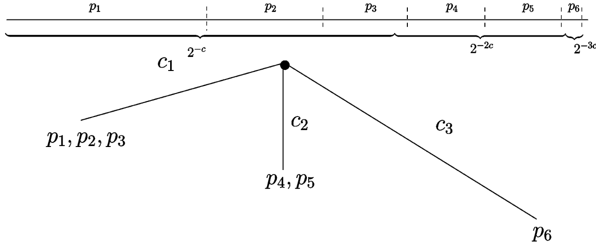

Fig. 3. The first splitting step for a case when $n = 6, c_1 = 1, c_2 = 2, c_3 = 3$, and the associated preliminary tree. This step groups p_1, p_2, p_3 as the first group, p_4, p_5 as the second and p_6 by itself. Note that we have not yet formally explained *why* we have grouped the items this way.

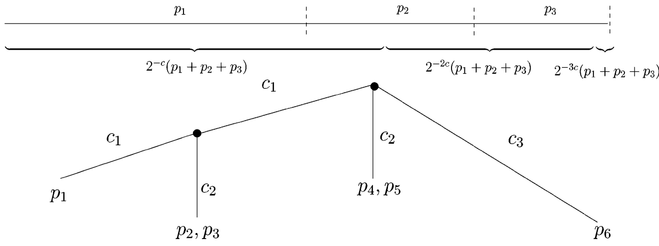

Fig. 4. In the second split, p_1 is kept by itself and p_2, p_3 are grouped together.

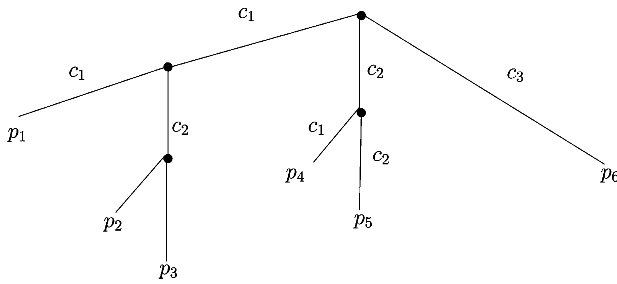

Fig. 5. After two more splits, the final coding tree is constructed. The associated code is $\{\sigma_1\sigma_1, \sigma_1\sigma_2\sigma_1, \sigma_1\sigma_2\sigma_2, \sigma_2\sigma_1, \sigma_2\sigma_2, \sigma_3\}$.

generating function rules, see, e.g., [27], state that $L(z) = (1 - D(z))^{-1}$. Observe that $l_n = 0$ if n is odd and $l_n = \binom{n}{n/2}$ for n even, so

$$L(z) = \sum_{n=0}^{\infty} \binom{2n}{n} (z^2)^n = \frac{1}{\sqrt{1-4z^2}}$$

and

$$\sum_{n=1}^{\infty} d_j z^j = D(z) = 1 - \sqrt{1-4z^2}.$$

This can then be solved to see that, for even $n > 0$, $d_n = 2C_{n/2-1}$ where $C_i = \frac{1}{i+1}\binom{2i}{i}$ is the ith Catalan number. For $n = 0$ or odd n, $d_n = 0$.

III. Notations and Definitions

There is a very standard correspondence between prefix-free codes over alphabet Σ and $|\Sigma|$-ary trees in which the mth child of node v is labeled with character $\sigma_m \in \Sigma$. A path from the root in a tree to a leaf will correspond to the word constructed by reading the edge labels while walking the path. The tree T corresponding to code $C = \{w_1, \ldots, w_n\}$ will be the tree containing the paths corresponding to the respected words. Note

```
CODE(l, r, U);
{Constructs codewords U_l, U_{l+1}, ..., U_r for p_l, p_{l+1}, ..., p_r.
 U is previously constructed common prefix of U_l, U_{l+1}, ..., U_r.}
If l = r
    then codeword U_l is set to be U.
else   {Distribute p_i s into initial bins I*_m}
    L = P_{l-1}; R = P_r; w = R - L
    ∀m, let L_m = L + w Σ_{i=1}^{m-1} 2^{-c c_i} and R_m = L_m + w 2^{-c c_m}.
        set I*_m = {k | L_m ≤ s_k < R_m} }
    {Shift the bins to become final I_m. Afterwards,
     all bins > M are empty, all bins ≤ M non-empty
     and ∀m ≤ M, I_m = {l_m, ... r_m}}

    {shift left so there are no empty "middle" bins.}
    M = 0; k = l;
    while k ≤ r do
        M = M + 1;
        l_M = k; r_M = max({k} ∪ {i > k | i ∈ I*_M});
        k = r_M + 1;

    {If all p_i's are in first bin, (right) shift p_r to 2^nd bin }
    if r_1 = r then
        M = 2;
        r_1 = r - 1; l_2 = r_2 = r;

    for m = 1 to M do
        CODE(l_m, r_m, U σ_m);
```

Fig. 6. Our algorithm. Note that the first step of creating the I^*_m was written to simplify the development of the analysis. In practice, it is not needed since I^*_m is only used to find $\max\{i > k \mid i \in I^*_m\}$ and this value can be calculated using binary search at the time it is required.

that the leaves in the tree will then correspond to codewords while internal nodes will correspond to prefixes of codewords. See Figs. 2 and 5.

Because this correspondence is one-to-one, we will speak about codes and trees interchangeably, with the cost of a tree being the cost of the associate code.

Definition 1: Let C be a prefix-free code over Σ and T its associated tree. N_T will denote the set of *internal nodes* of T.

Definition 2: Set c to be the unique positive solution to $1 = \sum_{i=1}^{t} 2^{-cc_i}$. Note that if $t < \infty$, then c must exists while if $t = \infty$, c might not exist. We only define c for the cases in which it exists. c is sometimes called the *root of the characteristic equation* of the letter costs.

Definition 3: Given letter costs c_i and their associated characteristic root c, let T be a code with those letter costs. If $p_1, p_2, \ldots, p_n \geq 0$ is a probability distribution then the *redundancy* of T relative to the p_i is

$$R(T; p_1, \ldots, p_n) = C(T) - \frac{1}{c} H(p_1, \ldots, p_n).$$

We will also define the *normalized redundancy* to be

$$\mathrm{NR}(T; p_1, \ldots, p_n) = cR = cC(T) - H(p_1, \ldots, p_n).$$

If the p_i and T are understood, we will write $R(T)$ ($\mathrm{NR}(T)$) or even R (NR).

We note that many of the previous results in the literature, e.g., (3) from [4], were stated in terms of NR. We will see later that this is a very natural measure for deriving bounds. Also, note that by the lower bound previously mentioned,

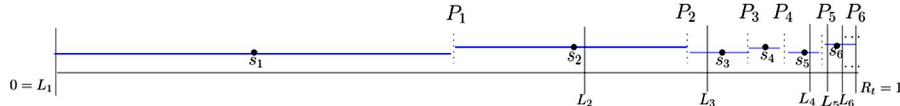

Fig. 7. The first step in our algorithm's splitting procedure. $n=6$. $L_i=\sum_{m=1}^{i-1} 2^{-cc_m}$. Note that even though only the first five L_i are shown, there might be an infinite number of them (if $t=\infty$). Note too that, for $0<i$, $L_i=R_{i-1}$.

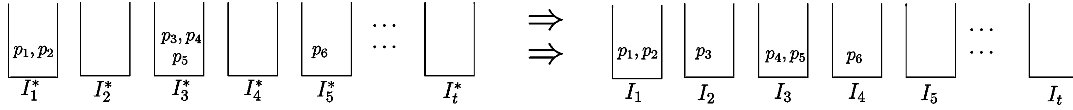

Fig. 8. The splitting procedure performed on the above example creates the bins I_m^* on the left. The shifting procedure then creates the I_m on the right.

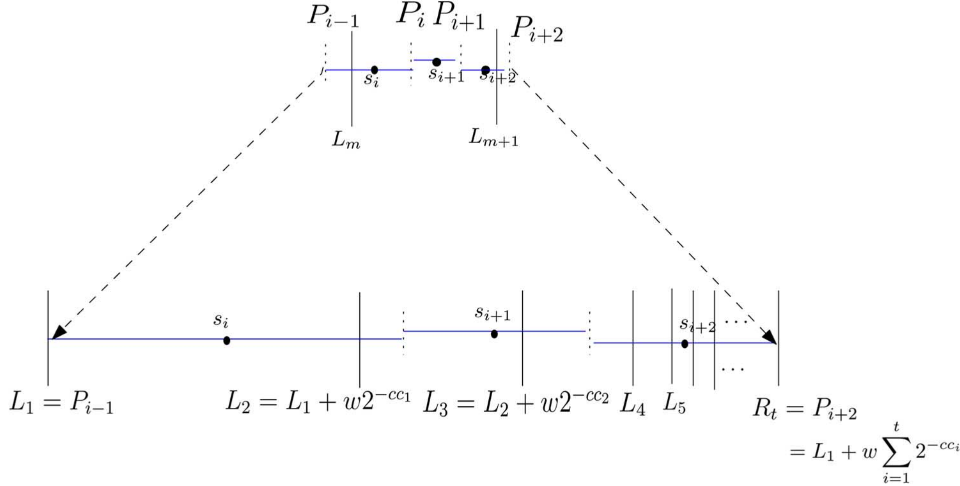

Fig. 9. An illustration of the recursive step of the algorithm. p_i, p_{i+1}, p_{i+2} have been grouped together. In the next splitting step, the interval operated on has length $w=p_i+p_{i+1}+p_{i+2}$.

$C(T) \geq \frac{1}{c}H(p_1,\ldots,p_n)$ for all T and p_i, so $R(T;p_1,\ldots,p_n)$ is a good measure of absolute error.

IV. The Algorithm

All of the provably efficient heuristics for the problem, e.g., [3], [4], [19]–[21], use the same basic approach, which itself is a generalization of Shannon's original binary splitting algorithm [28]. The idea is to create t *bins*, where bin m has weight 2^{-cc_m} (so the sum of all bin weights is 1). The algorithms then try to partition the probabilities into the bins; bin m will contain a set of contiguous probabilities $p_{l_m}, p_{l_m+1},\ldots,p_{r_m}$ whose sum will have total weight "close" to 2^{-cc_m}. The algorithms fix the first letter of all the codewords associated with the p_k in bin m to be σ_m. After fixing the first letter, the algorithms then recurse, normalizing $p_{l_m}, p_{l_m+1},\ldots,p_{r_m}$ to sum to 1, taking them as input and starting anew. The various algorithms differ in how they group the probabilities and how they recurse. See Figs. 3–5 for an illustration of this generic procedure.

Here we use a generalization of the version introduced in [4]. The algorithm first preprocesses the input and calculates all $P_k=p_1+p_2+\cdots+p_k$ ($P_0=0$) and $s_k=p_1+p_2+\cdots+p_{k-1}+\frac{p_k}{2}$. Note that if we lay out the p_i along the unit interval in order, then s_k can be seen as the *midpoint* of *interval* p_i. The algorithm then partitions the probabilities into ranges, and for each range it constructs left and right boundaries L_m, R_m. p_k will be assigned to bin m if, as an interval, p_k "falls" into the "range" $[L_m, R_m)$.

If the interval p_k completely falls into the range, i.e., $L_m \leq P_{k-1} < P_k < R_m$ then p_k should definitely be in bin m. But what if p_k spans two (or more) ranges, e.g., $L_m \leq P_{k-1} < R_m < P_k$? To which bin should p_k be assigned? The choice made by [4] is that p_k is assigned to bin m if $s_k=p_1+p_2+\cdots+p_k/2$ falls into $[L_m,R_m)$, i.e., the midpoint of p_k falls into the range.

Our procedure $CODE(l,r,U)$ will build a prefix-free code for $p_l,\ldots,p_r$ in which every codeword starts with prefix U. To build the entire code we call $CODE(1,n,\lambda)$, where λ is the empty string.

The procedure works as follows (Fig. 6 gives pseudocode and Figs. 7–9 illustrate the concepts)

Assume that we currently have a prefix of U assigned to $p_l,\ldots,p_r$. Let v be a node in the tree associated with U. Let $w(v)=\sum_{k=l}^{r} p_k$.

i) If $l=r$ then word U is assigned to p_l. Correspondingly, v is a leaf in the tree with weight $w(v)=p_l$.
ii) Otherwise let $L=P_{l-1}$ and $R=P_r$. Split $R-L=w(v)$ into t ranges[4] as follows. $\forall 1\leq m\leq t$

$$L_m = L+(R-L)\sum_{i=1}^{m-1} 2^{-cc_i}$$

$$R_m = L+(R-L)\sum_{i=1}^{m} 2^{-cc_i}.$$

Insert p_k, $l\leq k\leq r$ in bin m if $s_k\in[L_m,R_m)$. Bin m will thus contain the p_k in $I_m^*(v)=\{k \mid L_m\leq s_k<R_m\}$.

[4]In the description, t is permitted to be finite or infinite.

We now shift the items p_k *leftward* in the bins as follows. Walk through the bins from left to right. If the current bin already contains some p_k, continue to the next bin. If the current bin is empty, take the first p_k that appears in a bin to the right of the current one, shift p_k into the current bin, and walk to the next bin. Stop when all p_k have been seen. Let $I_m(v)$ denote the items in the bins after this shifting.

We note that it is not necessary to actually construct the $I^*_m(v)$ first. We only did so because viewing the construction as a two-stage procedure of first finding the $w^*(v)$ and then the $w(v)$ will be useful in our later analysis. Our main reason for introducing the left shift is that, after it is completed, there will be some $M(v)$ such that all bins $m \le M(v)$ will be nonempty and all bins $m > M(v)$ empty.

This observation permits constructing the $I_m(v)$ from scratch by walking from left to right, using a binary search each time, to find the rightmost item that should be in the current bin. This will take $O(M(v)\log(l-r))$ time in total.

We then check if all of the items are in $I_1(v)$. If they are, we take p_r and move it into $I_2(v)$ (and set $M(v)=2$). We call this a *right-shift*. Note that left-shifts and right-shifts cannot both occur while processing the same node.

Finally, after creating all of the $I_m(v)$, we let $l_m = \min\{k \in I_m(v)\}$ and $r_m = \max\{k \in I_m(v)\}$ and recurse, for each $m < M(v)$ building $CODE(l_m, r_m, U\sigma_m)$.

It is clear that the algorithm builds some prefix code with associated tree T. As defined, let N_T be the set of internal nodes of T. Since every internal node of T has at least two children, $\sum_{v\in N_T} M(v) \le 2n-1$.

The algorithm uses $O(1)$ time at each of its n leaves and $O(M(v)\log n)$ time at node v. Its total running time is thus bounded by

$$n + \sum_{v\in N_T} \log n M(v) = O(n\log n)$$

with no dependence upon t.

For comparison, we point out the algorithm in [4] also starts by first finding the $I^*_m(v)$. Since it assumed $t<\infty$, its shifting stage was much simpler, though. It just shifted p_l into the first bin and p_r into the tth bin (if they were not already there). This can, using an amortization technique from [21], be implemented in $\Theta(tn)$ time. Note the explicit dependence upon t, which is not permissible in the $t=\infty$ case.

We will now see that our modified shifting procedure not only permits a finite algorithm for infinite encoding alphabets, but also often provides a provably better approximation for *finite* encoding alphabets.

V. Analysis

In the analysis we define $w^*_m(v) = \sum_{k\in I^*_m(v)} p_k$, $w_m(v) = \sum_{k\in I_m(v)} p_k$. Note that

$$w(v) = \sum_{m=1}^{t} w^*_m(v) = \sum_{m=1}^{t} w_m(v) = \sum_{k=l}^{r} p_k.$$

Also, unless $t<\infty$ is specifically noted, t is permitted to be infinite.

Our starting point is three lemmas from [4]. The first was proven by recursion on the nodes of a tree and the second followed from the definition of the splitting procedure. The third was stated as a series of observations that could be derived from the second via some straightforward algebraic manipulations. Since our shifting procedure forces us to use a slightly different notation than [4], we restate the lemmas using our notation. Also, since [4] implicitly assumes that $t<\infty$, we follow every lemma with a small note explaining why the proof does/doesnot extend to the case $t=\infty$ and other issues that might affect our analysis.

Lemma 1: [4] Let $\bar{T}$ be a code tree and $N_{\bar{T}}$ be the set of all *internal* nodes of $\bar{T}$. Let $\bar{w}(v)$ and $\bar{w}_m(v)$ be the associated weights at the nodes and children of the nodes.

1) The cost $C(\bar{T})$ of the code tree $\bar{T}$ is

$$C(\bar{T}) = \sum_{v\in N_{\bar{T}}} \sum_{m=1}^{t} c_m \cdot \bar{w}_m(v).$$

2) The entropy $H(p_1,p_2,\dots,p_n)$ is

$$H(p_1,p_2,\dots,p_n) = \sum_{v\in N_{\bar{T}}} \bar{w}(v)\cdot H\left(\frac{\bar{w}_1(v)}{\bar{w}(v)}, \frac{\bar{w}_2(v)}{\bar{w}(v)}, \dots\right).$$

Note: As proven in [4], this lemma is valid for any code tree that contains a finite number of edges. The proof never uses the finiteness of t. Also, [4] states the lemma particularly for what we call $w^*(v)$ and $w^*_m(v)$. The proof, though, *only* uses the fact that these values are defined by the distribution of weights on a code tree and never uses any facts about *how* the code tree was created. We therefore state the lemma in its full generality using $\bar{w}(v)$ and $\bar{w}_m(v)$.

This lemma is valid for *any* code tree. In particular, we can apply it to express the normalized redundancy of the T built by our algorithm as

$$\begin{aligned} NR(T) &= c\cdot C(T) - H(p_1,p_2,\dots,p_n) \\ &= \sum_{v\in N_T} w(v)\left[\sum_{m=1}^{t} \frac{w_m(v)}{w(v)}\left(\log 2^{cc_m} + \log\frac{w_m(v)}{w(v)}\right)\right]. \end{aligned}$$

Set

$$E(v,m) = \frac{w_m(v)}{w(v)}\left(\log 2^{cc_m} + \log\frac{w_m(v)}{w(v)}\right).$$

Note that $\mathrm{NR}(T) = \sum_{v\in N_T} w(v)(\sum_{m=1}^{t} E(v,m))$. For convenience we will also define

$$E^*(v,m) = \frac{w^*_m(v)}{w(v)}\left(\log 2^{cc_m} + \log\frac{w^*_m(v)}{w(v)}\right),$$

$$\mathrm{NR}^*(T) = \sum_{v\in N_T} w(v)\left(\sum_{m=1}^{t} E^*(v,m)\right).$$

The analysis proceeds by bounding the values of $\mathrm{NR}^*(T)$ and $\mathrm{NR}(T) - \mathrm{NR}^*(T)$.

Lemma 2: [4]
(*Note:* In this lemma, the p_i can be arbitrarily ordered.)

Consider any call $CODE(l,r,U)$ with $l < r$. Let node v correspond to the word U. Let sets $I_1^*, I_2^*, \ldots$ be defined as in procedure CODE.

(a) If $I_m^* = \emptyset$, then $w_m^*(v) = 0$.

(b) If $I_m^* = \{e\}$, then $w_m^*(v) = p_e$.

(c) If $|I_m^*| \geq 2$. Let $e = \min I_m^*$ and $f = \max I_m^*$.

i) If $2 \leq m < t$, then

$$\frac{w_m^*(v)}{w(v)} \leq 2^{-cc_m} + \frac{p_e + p_f}{2w(v)} \leq 2 \cdot 2^{-cc_m}.$$

ii) If $m = 1$, then

$$\frac{w_m^*(v)}{w(v)} \leq 2^{-cc_1} + \frac{p_f}{2w(v)} \leq 2 \cdot 2^{-cc_1}.$$

.

iii) If $m = t$ (note that this case requires $t < \infty$) then

$$\frac{w_t^*(v)}{w(v)} \leq 2^{-cc_t} + \frac{p_e}{2w(v)} \leq 2 \cdot 2^{-cc_t}$$

Note: The proof of this lemma is local. It assumes that the call $CODE(l,r,U)$ is given and only concerns itself with partitioning the p_i associated with U. It never uses any information about *how* the call was arrived at, i.e., the rest of the tree. Thus, the fact that we perform an extra shifting procedure before making the call does not change the analysis. Also, the proofs of cases (a), (b), and c i),ii)) never use the finiteness of t.

Lemma 3: [4]
(*Note:* In this lemma, the p_i can be arbitrarily ordered.)

In case (c) of Lemma 2, $E^*(v,m) \leq \frac{p_e+p_f}{w(v)}$. Furthermore, if $m = 1$ then $E^*(v,m) \leq \frac{p_f}{w(v)}$, while if $m = t < \infty$, then $E^*(v,m) \leq \frac{p_e}{w(v)}$.

Note: This lemma follows from the previous ones by some straightforward algebraic manipulations that never use the fact that $t < \infty$.

As mentioned, since we are only interested in general and not alphabetic coding, we may take the p_i in any arbitrary order we like. In particular, Lemma 3 implies the following.

Corollary 4: If the p_i are sorted in nonincreasing order then in case (c) of Lemma 2

$$\text{if } m = 1, \text{then } E^*(v,m) \leq \frac{p_f}{w(v)}, \qquad \text{while if } m > 1, \text{then } E^*(v,m) \leq \frac{2p_e}{w(v)}.$$

We can now prove the technical lemma which is the basis of most of our results. This lemma explicitly uses the facts that the p_i are nonincreasing and that the c_i are nondecreasing to bound the error that can result from the left and right shifts performed by the algorithm of Fig. 6.

Lemma 5:

$$\mathrm{NR} - \mathrm{NR}^* \leq c(c_2 - c_1)\sum_{i \in A} p_i$$

where

$$A = \{i \,|\, i \text{ is right shifted by the algorithm at some step}\}.$$

Note: p_1 can never be right shifted, so $\sum_{i\in A} p_i \leq 1 - p_1$.

Proof: Define

$$X(v) = \sum_{m=1}^{t} w(v)E(v,m)$$

and

$$X^*(v) = \sum_{m=1}^{t} w(v)E^*(v,m)$$

Note that $\mathrm{NR} = \sum_{v\in N_T} X(v)$ and $\mathrm{NR}^* = \sum_{v\in N_T} X^*(v)$ For each v we will compare $X^*(v)$ and $X(v)$. If no shifts were performed while processing v, then $X^*(v) = X(v)$ and there is nothing to do. We now examine the two mutually exclusive cases of performing left shifts or performing a right shift.

Left shifts:

Every step in our left-shifting procedure involves taking a probability out of some bin m and moving it into some currently empty bin $r < m$. Let $w'_m(v)$ be the weight in bin m before that shift and p be the probability of the item being shifted.[5] Note that the original weight of bin r was $w'_r(v) = 0$ while after the shift, bin r will have weight p and bin m weight $w'_m(v) - p$. We use the trivial fact

$$\forall p, q > 0, \quad p\log p + q\log q \leq (p+q)\log(p+q). \tag{5}$$

Setting $q = w'_m(v) - p$ in (5) implies

$$p\log\frac{p}{w(v)} + (w'_m(v) - p)\log\frac{(w'_m(v) - p)}{w(v)} \leq w'_m(v)\log\frac{w'_m(v)}{w(v)}.$$

Furthermore, the fact that the c_i are nondecreasing implies

$$p\log 2^{cc_r} + (w'_m(v) - p)\log 2^{cc_m} \leq w'_m(v)\log 2^{cc_m}. \tag{6}$$

Combining the two last equations gives that

$$p\left(\log 2^{cc_r} + \log\frac{p}{w(v)}\right) + (w'_m(v) - p)\left(\log 2^{cc_m} + \log\frac{w'_m(v) - p}{w(v)}\right) \leq w'_m(v)\left(\log 2^{cc_m} + \log\frac{w'_m(v)}{w(v)}\right),$$

Since moving from $X^*(v)$ to $X(v)$ involves only operations in which probabilities are shifted to the left into an empty bucket, the analysis above implies that $X(v) \leq X^*(v)$.

[5] To be clear; since we are examining intermediate stages of the shifting procedure, it is possible that $w'(v)$ $(w'_m(v))$ is not equal to either $w^*(v)$ or $w(v)$ $(w_m^*(v)$ or $w_m(v))$.

Note: The calculations above show that "left-shifting" does not increase the code cost. They also imply that it might not improve it, either. So, why include it? The reason for left-shifting was *not* to reduce the code cost. It was, as described at the end of Section IV, to remove any dependence on t from the running time of the algorithm.

Right shifts:
Consider node v. Suppose that all of the probabilities in v fall into I_1^* with $I_1^* = \{p_e, \dots, p_f\}$ and $e \neq f$. Since p_f starts in bin 1, p_e must be totally contained in bin 1, so $p_e \leq 2^{-cc_1} w(v)$. The algorithm shifts p_f to the right giving $I_1 = I_1 - \{p_f\}$ and $I_2 = \{p_f\}$. The p_i are nonincreasing so $p_f \leq p_e$

$$\begin{aligned} E(v,2) &= \frac{p_f}{w(v)} \left(\log 2^{cc_2} + \log \frac{p_f}{w(v)} \right) \\ &\leq \frac{p_f}{w(v)} (cc_2 - cc_1). \end{aligned}$$

Also, $E(v,1) \leq E^*(v,1)$. Thus

$$\begin{aligned} w(v) \sum_{m=1}^{t} E(v,m) &= w(v)E(v,1) + w(v)E(v,2) \\ &\leq w(v)E^*(v,1) + p_f(cc_2 - cc_1) \end{aligned}$$

Once a p_f is right-shifted it immediately becomes a leaf and can never be right-shifted again.

Combining the analyses of left shifts and right shifts gives

$$\begin{aligned} \text{NR} = \sum_{v \in N_T} X(v) &\leq \sum_{v \in N_T} X^*(v) + c(c_2 - c_1) \sum_{i \in A} p_i \\ &= \text{NR}^* + c(c_2 - c_1) \sum_{i \in A} p_i. \end{aligned}$$

□

Lemma 6:

$$\text{NR}^* \leq 2(1-p_1) + \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v)E^*(v,m).$$

Proof: We evaluate NR^* by partitioning it into

$$\begin{aligned} \text{NR}^* = &\sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)| \geq 2}} w(v)E^*(v,m) \\ &+ \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v)E^*(v,m). \quad (7) \end{aligned}$$

We use a generalization of an amortization argument developed in [4] to bound the first summand. From Corollary 4, we know that if $|I_m^*(v)| \geq 2$ with $e = \min I_m^*$ and $f = \max I_m^*$ then $w(v)E^*(v,m)$ is at most (a) p_f or (b) $2p_e$, depending upon whether (a) $m = 1$, or (b) $m > 1$.

Suppose that some p_i appears as $2p_i$ in such a bound because $i = \min I_m^*(v)$, i.e., case (b). Then, in all later recursive steps of the algorithm i will always be the leftmost item in bin 1 and will therefore not be used in any later case (a) or (b) bound.

Now suppose that some p_i appears in such a bound because $i = \max I_m^*(v)$, i.e., case (a). Then in all later recursive steps of the algorithm, i will always be the rightmost item in the rightmost nonempty bin. The only possibility for it to be used in a later bound is if becomes the rightmost item in bin 1, i.e., all of the probabilities are in $I_1^*(v)$. In this case, p_i is used for a second case (a) bound. Note that if this happens, then p_i is immediately right-shifted, becomes a leaf in bin 2, and is never used in any later recursion.

Any given probability p_i can therefore be used either once as a case (b) bound and contribute $2p_i$ or twice as a case (b) bound and again contribute $2p_i$. Furthermore, p_1 can never appear in a case (a) or (b) bound because, until it becomes a leaf, it can only be the leftmost item in bin 1. Thus

$$\sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)| \geq 2}} w(v)E^*(v,m) \leq 2(1-p_1). \quad (8)$$

□

Note: In Mehlhorn's original proof [4] the value corresponding to the right-hand side (RHS) of (8) was $(1-p_1-p_n)$. This is because the shifting step of Mehlhorn's algorithm guaranteed that $|I_t^*(v)| \neq 0$ and thus there was a symmetry between the analysis of leftmost and rightmost bins. In our situation, t might be infinity so we cannot assume that the rightmost nonempty bin is t and we get $2(1-p_1)$ instead.

Combining this Lemma with Lemma 5 gives the following.

Corollary 7:

$$\begin{aligned} \text{NR} \leq 2(1-p_1) + c(c_2 - c_1) &\sum_{i \in A} p_i \\ &+ \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v)E^*(v,m). \end{aligned}$$

We will now see different bounds on the last summand in the above expression. Section VI compares the results we get to previous ones for different classes of $\mathcal{C}$. Before proceeding, we note that any p_i can only appear as $I_m^*(v) = \{p_i\}$ for at most one (m,v) pair. Furthermore, if p_i does appear in such a way, then it cannot have been made a leaf by a previous right shift and thus $p_i \notin A$.

We start by noting that, when $t \leq \infty$, our bound is never worse than 1 plus the old bound of $(1-p_1-p_n) + cc_t$ stated in (3).

Theorem 1: If $t < \infty$ then

$$\text{NR} \leq 2(1-p_1) + cc_t.$$

Proof: If $I_m^*(v) = \{p_i\}$ then $w(v)E^*(v,m) \leq p_i cc_m$ so

$$\begin{aligned} \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v)E^*(v,m) &\leq \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} p_i cc_m \\ &\leq cc_t \sum_{i \notin A} p_i. \end{aligned}$$

The theorem then follows from Corollary 7. □

For a tighter analysis we will need a better bound.

Lemma 8:

(a) Let $v \in N_T$. Suppose i is such that $i \in I_m^*(v)$. Then

$$\frac{p_i}{w(v)} \leq 2 \cdot \sum_{j=m}^{t} \frac{1}{2^{c \cdot c_j}}.$$

(b) Further, suppose there is some $m' > m$ such that $I^*_{m'} \neq \emptyset$. Then

$$\frac{p_i}{w(v)} \leq 2 \cdot \sum_{j=m}^{m'} \frac{1}{2^{c \cdot c_j}} \leq 4 \cdot \sum_{j=m}^{m'-1} \frac{1}{2^{c \cdot c_j}}.$$

Proof: Consider the call $CODE(l, r, U)$ at node v. The fact that $i \in I^*_m(v)$ implies $L + \sum_{j=1}^{m-1} 2^{-cc_j} = L_m \leq s_i$. To prove (a) just note that

$$s_i + \frac{p_i}{2} = P_i \leq P_r = R = L + w(v) \sum_{i=1}^{t} 2^{-cc_i}.$$

So $\frac{p_i}{2} \leq w(v) \sum_{j=m}^{t} \frac{1}{2^{c \cdot c_j}}$.

To prove part (b) let $i' \in I^*_{m'}$. Then

$$s_i + \frac{p_i}{2} = P_i \leq s_{i'} < L + w(v) \sum_{j=1}^{m'} 2^{-cc_j}.$$

So $\frac{p_i}{2} \leq w(v) \sum_{j=m}^{m'} \frac{1}{2^{c \cdot c_j}}$. The final inequality follows from the fact that $c_{m'-1} \leq c_m$. □

Definition 4: Set $\beta_m = 2^{cc_m} \sum_{i=m}^{t} 2^{-cc_i}$ and $\beta = \sup\{\beta_m \mid 1 \leq m \leq t\}$.

Note: This definition is valid for both $t < \infty$ and $t = \infty$.

We can now prove our first improved bound.

Theorem 2: If $\beta < \infty$ then

$$\mathrm{NR} \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log \beta).$$

Proof: Recall that

$$w(v) E^*(m, v) = w^*_m(v) \left(\log 2^{cc_m} + \log \frac{w^*_m(v)}{w(v)} \right).$$

In the special case that $|I^*_m(v)| = 1$, i.e., $I^*_m(v) = \{i\}$ for some i, Lemma 8(a) tells us that

$$\frac{w^*_m(v)}{w(v)} = \frac{p_i}{w(v)} \leq 2 \sum_{i=m}^{t} 2^{-cc_i}.$$

Using the above and Definition 4 we can bound the last summands in Corollary 7 as

$$\begin{aligned} w(v) E^*(m, v) &= w^*_m(v) \left(\log 2^{cc_m} + \log \frac{w^*_m(v)}{w(v)} \right) \\ &\leq w^*_m(v) \log \left(2^{cc_m} 2 \sum_{i=m}^{t} 2^{-cc_i} \right) \\ &\leq w^*_m(v)(1 + \log \beta). \end{aligned}$$

If $I^*_m(v) = \{i\}$ then i was not a leaf in any previous step and therefore could not have been right-shifted, so $i \notin A$. Thus

$$\sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I^*_m(v)| = 1}} w(v) E^*(v, m) \leq (1 + \log \beta) \sum_{i \notin A} p_i. \quad \square$$

This immediately gives an improved bound for many finite cases because, if $t < \infty$, then $\beta_m = 2^{cc_m} \sum_{i=m}^{t} 2^{-cc_i} \leq t - m + 1$ so $\beta \leq t$. Thus we have the following.

Theorem 3: If t is finite then

$$\mathrm{NR} \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log t)$$

Definition 5: For all $j \geq 1$, set

$$d_j = |\{i \mid j \leq c_i < j + 1\}|.$$

This permits us to give another general bound that also works for many infinite alphabets.

Lemma 9: If $d_j = O(1)$, then $\mathrm{NR} = O(1)$. In particular, if $\forall j,\, d_j \leq K$ then $\beta \leq \frac{2^c K}{1 - 2^{-c}}$ so, from Theorem 2

$$\mathrm{NR} \leq 2(1 - p_1) + \max\left(c(c_2 - c_1),\ 1 + c + \log\left(\frac{K}{1 - 2^{-c}} \right) \right).$$

Furthermore, if all of the c_i are integers, then $\beta \leq \frac{K}{1 - 2^{-c}}$ and

$$\mathrm{NR} \leq 2(1 - p_1) + \max\left(c(c_2 - c_1), 1 + \log\left(\frac{K}{1 - 2^{-c}} \right) \right).$$

Proof: Since $c_1 = 1$ we must have $2^{-c} < 1$. Thus, for all $m \geq 1$, if $\ell \leq c_m < \ell + 1$ then

$$\begin{aligned} \beta_m &= 2^{cc_m} \sum_{i=m}^{t} 2^{-cc_m} \\ &\leq 2^{c(\ell+1)} \sum_{j=\ell}^{\infty} d_j 2^{-cj} \\ &\leq 2^c K 2^{c\ell} \sum_{j=\ell}^{\infty} 2^{-cj} = \frac{2^c K}{1 - 2^{-c}} \end{aligned}$$

which is independent of m and ℓ. The analysis when the c_i are all integers is similar. □

For general infinite alphabets we are not able to derive a constant redundancy bound but we can prove.

Theorem 4: If $\mathcal{C}$ is infinite and $\sum_{m=1}^{\infty} c_m 2^{-cc_m} < \infty$, then, for every $\epsilon > 0$

$$R \leq \epsilon \frac{1}{c} H(p_1, \ldots, p_n) + f(\mathcal{C}, \epsilon) \tag{9}$$

where $f(\mathcal{C}, \epsilon)$ is some constant based only on $\mathcal{C}$ and ϵ. Note that this is equivalent to stating that

$$C(T) \leq (1 + \epsilon) OPT + f(\mathcal{C}, \epsilon)$$

Proof: We must bound the

$$\sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I^*_m(v)| = 1}} w(v) E^*(v, m)$$

term from the RHS of Corollary 7. Recall that $|I^*_m(v)| = 1$ means that $\exists i$ such that $I^*_m(v) = \{i\}$, i.e., $w^*_m(v) = p_i$ and thus $w(v) E^*(v, m) \leq p_i c c_m$.

For every $\epsilon > 0$, we associate a value N_ϵ (to be determined later) and set $m_\epsilon = \max\{m \mid c_m \le N_\epsilon\}$. Since no probability appears more than once in the sum we can write

$$\sum_{v \in N_T} \sum_{\substack{1 \le m \le m_\epsilon \\ |I^*_m(v)|=1}} w(v) E^*(v,m) \le cN_\epsilon.$$

To analyze the remaining cases, fix v. Consider the set of indices

$$M_v = \{m \mid (m > m_\epsilon) \text{ and } |I^*_m(v) = 1|\}.$$

Sort these indices in increasing order so that $M_v = \{m_1, m_2, \ldots, m_r\}$ for some r with $m_1 < m_2 < \cdots < m_r$. Let i_j be such that $I^*_{m_j}(v) = \{p_{i_j}\}$. Thus

$$\sum_{\substack{m_\epsilon < m \\ |I^*_m(v)|=1}} w(v) E^*(v,m) = \sum_{j=1}^{r} w(v) E^*(v, m_j) \le \sum_{j=1}^{r} p_{i_j} c c_{m_j}.$$

Lemma 8 and the fact that the c_m are nondecreasing then gives

$$\begin{aligned} \sum_{j=1}^{r} p_{i_j} c c_{m_j} &\le cw(v) \left[\sum_{j=1}^{r-1} c_{m_j} \left(4 \sum_{m=m_j}^{m_{j+1}-1} 2^{-cc_m} \right) + 2c_{m_r} \sum_{m=m_r}^{\infty} 2^{-cc_m} \right] \\ &\le 4cw(v) \sum_{m=m_1}^{\infty} c_m 2^{-cc_m} \le 4cw(v) \sum_{m \ge m_\epsilon}^{\infty} c_m 2^{-cc_m}. \end{aligned}$$

We are given that $\sum_{m=1}^{\infty} c_m 2^{-cc_m}$ converges. Thus, $g(m_\epsilon) \downarrow 0$ as $m_\epsilon \to \infty$ where $g(x) = \sum_{m \ge x}^{\infty} c_m 2^{-cc_m}$.

Note that as N_ϵ increases, m_ϵ increases. Given ϵ, we now choose N_ϵ to be the smallest value such that $g(m_\epsilon) \le \frac{\epsilon}{8}$. Note that N_ϵ is independent of v.

Combine the above bounds as follows:

$$\begin{aligned} \sum_{v \in N_T} \sum_{\substack{1 \le m \le t \\ |I^*_m(v)|=1}} w(v) E^*(v,m) &= \sum_{v \in N_T} \sum_{\substack{1 \le m \le m_\epsilon \\ |I^*_m(v)|=1}} w(v) E^*(v,m) \\ &\quad + \sum_{v \in N_T} \sum_{\substack{m_\epsilon < m \\ |I^*_m(v)|=1}} w(v) E^*(v,m) \\ &\le cN_\epsilon + \sum_{v \in N_T} \frac{\epsilon}{2} cw(v) \end{aligned}$$

Recall from Lemma 1 and the fact that $\forall m, c_m \ge 1$,

$$\begin{aligned} C(T) &= \sum_{v \in N_T} \sum_{m=1}^{t} c_m \cdot w_m(v) \\ &\ge \sum_{v \in N_T} \sum_{m=1}^{t} w_m(v) = \sum_{v \in N_T} w(v). \end{aligned}$$

Thus, we have just seen that

$$\sum_{v \in N_T} \sum_{\substack{1 \le m \le t \\ |I^*_m(v)|=1}} w(v) E^*(v,m) \le cN_\epsilon + \frac{\epsilon}{2} cC(T).$$

Plugging back into Corollary 7 gives

$$\begin{aligned} &cC(T) - H(p_1, \ldots, p_n) \\ &\qquad \le 2(1-p_1) + c(c_2 - c_1) + cN_\epsilon + \frac{\epsilon}{2} cC(T) \end{aligned}$$

which can be rewritten as

$$\begin{aligned} &C(T) - \frac{1}{1-\frac{\epsilon}{2}} \frac{1}{c} H(p_1, \ldots, p_n) \\ &\qquad \le \frac{1}{1-\frac{\epsilon}{2}} \frac{1}{c} \left(2(1-p_1) + c(c_2 - c_1) + cN_\epsilon \right). \end{aligned}$$

We may assume that $\epsilon \le 1/2$, so $1 + \epsilon \ge \frac{1}{1-\frac{\epsilon}{2}}$. Thus

$$C(T) - (1+\epsilon) \frac{1}{c} H(p_1, \ldots, p_n) \le f(\mathcal{C}, \epsilon)$$

where

$$f(\mathcal{C}, \epsilon) = \frac{4}{3} \left(\frac{2}{c} + (c_2 - c_1) + N_\epsilon \right). \tag{10}$$

This can then be rewritten as

$$\begin{aligned} R = C(T) - \frac{1}{c} H(p_1, \ldots, p_n) &\le \epsilon \frac{1}{c} H(p_1, \ldots, p_n) + f(\mathcal{C}, \epsilon) \\ &\le \epsilon OPT + f(\mathcal{C}, \epsilon) \end{aligned}$$

proving the theorem. □

VI. EXAMPLES

We now examine some of the bounds derived in the last section and show how they compare to the old bound of $(1 - p_1 - p_n) + cc_t$ stated in (3). In particular, we show that for large families of costs the old bounds go to infinity while the new ones give uniformly constant bounds.

Case 1: $\mathcal{C}_\alpha = (c_1, c_2, \ldots, c_{t-1}, \alpha)$ with $\alpha \uparrow \infty$.
We assume $t \ge 3$ and all of the c_i, $i < t$, are fixed. Let $c^{(\alpha)}$ be the root of the corresponding characteristic equation $1 = 2^{-c\alpha} + \sum_{i=1}^{t-1} c^{-cc_i}$. Note that $c^{(\alpha)} \downarrow \bar{c}$ where $\bar{c}$ is the root of $1 = \sum_{i=1}^{t-1} c^{-cc_i}$. Let ($\text{NR}_\alpha$) R_α be the (normalized) redundancy corresponding to $\mathcal{C}_\alpha$.

For any fixed α, the old bound (3) would give

$$\text{NR}_\alpha \le (1 - p_1 - p_n) + c^{(\alpha)} \alpha, \quad R_\alpha \le \frac{(1 - p_1 - p_n)}{c^{(\alpha)}} + \alpha$$

the RHS of both of which tend to ∞ as α increases. Compare this to Theorem 3 which gives a uniform bound of

$$\begin{aligned} \text{NR}_\alpha &\le 2(1-p_1) + \max\Big(c^{(\alpha)}(c_2 - c_1), 1 + \log t \Big) \\ &\le 2(1-p_1) + \max\Big(c^{(c_{t-1})}(c_2 - c_1), 1 + \log t \Big) \end{aligned}$$

and

$$R_\alpha \le \frac{\text{NR}_\alpha}{c^{(\alpha)}} \le \frac{2(1-p_1) + \max\left(c^{(c_{t-1})}(c_2 - c_1), 1 + \log t \right)}{\bar{c}}.$$

For concreteness, we examine a special case of the above.

Example 1: Let $t = 3$ with $c_1 = c_2 = 1$ and $c_3 = \alpha \geq 1$. The old bounds (3) gives an asymptotically infinite error as $\alpha \to \infty$. The bound from Theorem 3 is

$$\mathrm{NR}_\alpha \leq 2(1-p_1)+\max\left(c^{(\alpha)}(c_2-c_1),\, 1+\log t\right) \leq 3+\log 3$$

independent of α. Since $c^{(\alpha)} \geq \bar{c} = 1$, we also get

$$R_\alpha = \frac{\mathrm{NR}_\alpha}{c^{(\alpha)}} \leq 3 + \log 3.$$

Case 2: A sequence of finite alphabets approaching an infinite one.

Let $\mathcal{C}$ be an infinite sequence of letter costs such that there exists a $K > 0$ satisfying for all j, $d_j = |\{i \,|\, j \leq c_i < j\}| \leq K$. Let c be the root of the characteristic equation $1 = \sum_{i=1}^{\infty} 2^{-cc_i}$. Let $\Sigma^{(t)} = \{\sigma_1, \ldots, \sigma_t\}$ and its associated letter costs be $\mathcal{C}^{(t)} = \{c_1, \ldots, c_t\}$. Let $c^{(t)}$ be the root of the corresponding characteristic equation $1 = \sum_{i=1}^{t} 2^{-cc_i}$ and (NR_t) R_t be the associated (normalized) redundancy. Note that $c^{(t)} \uparrow c$ as t increases.

For any fixed t, the old bound (3) would be $\mathrm{NR}_t \leq (1-p_1-p_n)+c^{(t)}c_t$ which goes to ∞ as t increases. Lemma 9 tells us that

$$\beta^{(t)} = \max_{1\leq m\leq t} 2^{c^{(t)}c_m} \sum_{i=1}^{t} 2^{c^{(t)}c_i} \leq \frac{2^c K}{1-2^{-c^{(t)}}} \leq \frac{2^c K}{1-2^{-c^{(2)}}}$$

so, from Theorem 2 and the fact that $\forall t,\, c^{(2)} \leq c^{(t)} < c$, we get

$$\mathrm{NR}_t \leq 2(1-p_1)+\max\left(c(c_2-c_1), 1+c+\log\frac{K}{1-2^{-c^{(2)}}}\right).$$

Note that if all of the c_m are integers, then the additive factor c will vanish.

Example 2: Let $\mathcal{C} = (1, 2, 3, \ldots)$. i.e., $c_m = m$. The old bounds (3) gives an asymptotically infinite error as $\alpha \to \infty$.

For this case $c = 1$ and $K = 1$. $c^{(2)}$ is the root of the characteristic equation $1 = 2^{-2} + 2^{-2c}$. Solving gives $2^{-c^{(2)}} = \frac{\sqrt{5}-1}{2}$ and $c^{(2)} = 1 - \log(\sqrt{5}-1) \approx 0.694\ldots$. Plugging into our equations gives

$$\begin{aligned}\mathrm{NR}_t &\leq 2(1-p_1)+\max\left(c(c_2-c_1),\, 1+\log\left(\frac{K}{1-2^{-c^{(2)}}}\right)\right)\\ &= 2(1-p_1)+1+\log\left(\frac{2}{3-\sqrt{5}}\right) \leq 4.388\end{aligned}$$

and

$$R_t = \frac{\mathrm{NR}_t}{c^{(t)}} \leq \frac{\mathrm{NR}_t}{c^{(2)}} \leq 6.232.$$

Case 3: An infinite case when $d_j = O(1)$.

In this case just apply Lemma 9 directly.

Example 3: Let $\mathcal{C}$ contain d copies each of $i = 1, 2, 3, \ldots$, i.e., $c_m = 1 + \lfloor \frac{m-1}{d} \rfloor$. Note that $K = d$. If $d = 1$, i.e., $c_m = m$, then $c = K = 1$ and

$$R = \mathrm{NR} \leq 2(1-p_1) + 2.$$

If $d > 1$, then $A(x) = \sum_{m=1}^{\infty} c_m z^m = \frac{dz}{1-z}$. The solution α to $A(\alpha) = 1$ is $\alpha = \frac{1}{d+1}$, so $c = -\log\alpha = \log(d+1)$. The lemma gives

$$\begin{aligned}\mathrm{NR} &\leq 2(1-p_1)+\left(1+\log\left(\frac{K}{1-2^{-c}}\right)\right) \leq 3+\log(d+1)\\ R &\leq 1+\frac{3}{\log(d+1)}.\end{aligned}$$

Case 4: d_j are integral and satisfy a linear recurrence relation. In this case, the generating function $A(z) = \sum_{j=1}^{\infty} d_j z^j = \sum_{m=1}^{\infty} z^{c_m}$ can be written as $A(z) = \frac{P(z)}{Q(z)}$ where $P(z)$ and $Q(z)$ are relatively prime polynomials. Let γ be a smallest modulus root of $Q(z)$. If γ is the unique root of that modulus (which happens in most interesting cases) then it is known that $d_j = \Theta(j^{d-1}\gamma^{-j})$ (which will also imply that γ is positive real) where d is the multiplicity of the root. There must then exist some $\alpha < \gamma$ such that $A(\alpha) = 1$. By definition $c = -\log\alpha$. Furthermore, since $\alpha < \gamma$ we must have that $\sum_{j=1}^{\infty} d_j j \alpha^j = \sum_{m=1}^{\infty} c_m \alpha^{c_m}$ also converges, so Theorem 4 applies.

Note that

$$\begin{aligned}h(x) &= \sum_{j=x}^{\infty} d_j j \alpha^j = O\left(\sum_{j=x}^{\infty} j^{d-1} j \left(\frac{\alpha}{\gamma}\right)^j\right)\\ &= O\left(x^d \left(\frac{\alpha}{\gamma}\right)^x\right)\end{aligned}$$

implying

$$h^{-1}(\epsilon) = \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon)$$

where we define

$$h^{-1}(\epsilon) = \max\{x \,|\, h(x) \leq \epsilon,\, h(x-1) > \epsilon\}.$$

Working through the proof of Theorem 4 we find that when the c_m are all integral

$$\begin{aligned}\forall m', \quad g(m') &= \sum_{m\geq m'} c_m 2^{-cc_m} = \sum_{m\geq m'} c_m \alpha^{c_m}\\ &\leq \sum_{j\geq c_{m'}} j d_j \alpha_j = h(c_{m'}).\end{aligned}$$

Recall that $m_\epsilon = \max\{m \,|\, c_m \leq N_\epsilon\}$. Then $g(m_\epsilon) \leq h(N_\epsilon)$. Since $g(m_\epsilon) \leq \epsilon/8$

$$N_\epsilon \leq h^{-1}(\epsilon/8) = \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon)$$

and thus our algorithm creates a code T satisfying

$$C(T) - OPT \leq \epsilon OPT + \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon). \quad (11)$$

Example 4: Consider the case where $d_j = F_j$, the jth Fibonacci number, $F_1 = 1$, $F_2 = 1$, $F_3 = 2, \ldots$. It is well known that

$$A(z) = \sum_{j=1}^{\infty} d_j z^j = \frac{x}{1-x-x^2}$$

and $F_j = \frac{\phi^n-(1-\phi)^n}{\sqrt{5}}$ where $\phi = \frac{1+\sqrt{5}}{2}$. Thus $d_j = \frac{\gamma^{-j}}{\sqrt{5}} + e_n$ where $\gamma = \phi^{-1}$ and $|e_n| < 1$. Solving $A(\alpha) = 1$ gives $\alpha = \sqrt{2}-1 \approx .4142\ldots$ (and $c = -\log\alpha = 1.2715\ldots$). Expression (11) gives a bound on the cost of the redundancy of our code with

$$\frac{\gamma}{\alpha} = \frac{2}{(1+\sqrt{5})(\sqrt{2}-1)} \approx 1.492\ldots.$$

Example 5: As discussed in Section II, Example 3 $d_j = F_j$ arises when modeling $\Sigma = \{1,2,3\}$ with associated $\mathcal{C} = (1,1,2)$; the problem there was to find minimal cost prefix-free codes in which all words end with a 1.

That problem was actually an illustration of the general Section II, Example 3 situation in which *finite* alphabet $\Sigma = \{\sigma_1,\ldots,\sigma_t\}$, associated cost vector $\mathcal{C} = (c_1,\ldots,c_t)$, $\Sigma' \subset \Sigma$, and $\mathcal{L} = \Sigma^*\Sigma'$ are given. We are then asked to find a minimum cost prefix-free code in $\mathcal{L}$. It was shown there that this can be modeled as an infinite alphabet problem in which the d_j satisfy a linear recurrence relation. Thus, all of these problems fit into the Case 4 framework.

Case 5: An example for which there is no known bound. An interesting open question is how to bound the redundancy for the case of balanced words described in Section II, Example 4. Recall that this had d_j integral with $d_j = 0$ for $j = 0$ and odd j and for even $j > 0$, $d_j = 2C_{j/2-1}$ where $C_i = \frac{1}{i+1}\binom{2i}{i}$ is the ith *Catalan number*. It is well known that $\sum_{j=0}^{\infty} C_j x^j = \frac{1}{2x}(1-\sqrt{1-4x})$ so

$$A(x) = \sum_{m=1}^{\infty} x^{c_m} = \sum_{j=1}^{\infty} d_j x^j = 1 - \sqrt{1-4x^2}.$$

Solving for $A(\alpha) = 1$ gives $\alpha = \frac{1}{2}$ and $c = -\log\alpha = 1$. On the other hand

$$\sum_{m=1}^{\infty} c_m x^{c_m} = \sum_{j=1}^{\infty} j d_j x^j = 2\sum_{j=1}^{\infty} \binom{2(j-1)}{j-1} (x^2)^j = \frac{x^2}{\sqrt{1-4x^2}}$$

so this sum *does not* converge when $x = 1/2$. Thus, we cannot use Theorem 4 to bound the redundancy. Some observation shows that this $\mathcal{C}$ does not satisfy any of our other theorems either. It remains an open question as to how to construct a code with "small" redundancy for this problem, i.e., a code with a constant additive approximation or something similar to Theorem 4.

VII. Conclusion and Open Questions

We have just seen $O(n \log n)$ time algorithms for constructing almost optimal prefix-free codes for source letters with probabilities $p_1,\ldots,p_n$ when the costs of the letters of the encoding alphabet are unequal values $\mathcal{C} = \{c_1, c_2, \ldots\}$. For many finite encoding alphabets, our algorithms have provably smaller redundancy (error) than previous algorithms given in [3], [4], [19], [21]. Our algorithms also are the first that give provably bounded redundancy for some infinite alphabets.

There are still many open questions left. The first arises by noting that, for the finite case, the previous algorithms were implicitly constructing *alphabetic codes*. Our proof explicitly uses the fact that we are only constructing general codes. It would be interesting to examine whether it is possible to get better bounds for alphabetic codes (or to show that this is not possible).

Another open question concerns Theorem 4 in which we showed that if $\sum_{m=1}^{\infty} c_m 2^{-cc_m} < \infty$, then

$$\forall\epsilon > 0, \quad C(T) - OPT \le \epsilon OPT + f(\mathcal{C},\epsilon).$$

Is it possible to improve this for some general $\mathcal{C}$ to get a purely additive error rather than a multiplicative one combined with an additive one?

Finally, in Case 5 of the last section we gave a natural example for which the root c of $\sum_{i=1}^{\infty} 2^{-cc_m} = 1$ exists but for which $\sum_{m=1}^{\infty} c_m 2^{-cc_m} = \infty$ so that we cannot apply Theorem 4 and therefore have no error bound. It would be interesting to devise an analysis that would work for such cases as well.

References

[1] D. A. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[2] I. Itai, "Optimal alphabetic trees," *SIAM J. Comput.*, vol. 5, pp. 9–18, 1976.

[3] R. M. Krause, "Channels which transmit letters of unequal duration," *Inf. Contr.*, vol. 5, pp. 13–24, 1962.

[4] K. Mehlhorn, "An efficient algorithm for constructing nearly optimal prefix codes," *IEEE Trans. Inf. Theory*, vol. IT-26, no. 5, pp. 513–517, Sep. 1980.

[5] N. M. Blachman, "Minimum cost coding of information," *IRE Trans. Inf. Theory*, vol. PGIT-3, no. 3, pp. 139–149, Mar. 1954.

[6] R. Marcus, "Discrete Noiseless Coding," M.S. thesis, Elec. eng. Dep., MIT, Cambridge, 1957.

[7] R. Karp, "Minimum-redundancy coding for the discrete noiseless channel," *IRE Trans. Inf. Theory*, vol. IT-7, no. 1, pp. 27–39, Jan. 1961.

[8] L. E. Stanfel, "Tree structures for optimal searching," *J. Assoc. Comp. Mach.*, vol. 17, no. 3, pp. 508–517, July 1970.

[9] B. Varn, "Optimal variable length codes (arbitrary symbol cost and equal code word probability)," *Inf. Contr.*, vol. 19, pp. 289–301, 1971.

[10] E. N. Gilbert, "How good is morse code," *Inf. Contr.*, vol. 14, pp. 585–565, 1969.

[11] E. N. Gilbert, "Coding with digits of unequal costs," *IEEE Trans. Inf. Theory*, vol. 41, no. 2, pp. 596–600, Mar. 1995.

[12] K. A. S. Immink, *Codes for Mass Data Storage Systems*. Rotterdam, The Netherlands: Shannon Foundation Pub., 1999.

[13] M. Golin and G. Rote, "A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs," *IEEE Trans. Inf. Theory*, vol. 44, no. 5, pp. 1770–1781, Sep. 1998.

[14] D. E. Knuth, *The Art of Computer Programming, Volume III: Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

[15] K. Hinderer, "On dichotomous search with direction-dependent costs for a uniformly hidden object," *Optimization*, vol. 21, no. 2, pp. 215–229, 1990.

[16] S. Kapoor and E. M. Reingold, "Optimum lopsided binary trees," *J. Assoc. Comput. Mach.*, vol. 36, no. 3, pp. 573–590, Jul. 1989.

[17] P. Bradford, M. Golin, L. L. Larmore, and W. Rytter, "Optimal prefix-free codes for unequal letter costs and dynamic programming with the monge property," *J. Algorithms*, vol. 42, pp. 277–303, 2002.

[18] M. J. Golin, C. Kenyon, and N. E. Young, "Huffman coding with unequal letter costs," in *Proc. 34th Annu. ACM Symp. Theory of Computing (STOC'02)*, Montreal, QC, Canada, May 2002, pp. 785–791.

[19] I. Csiszár, "Simple proofs of some theorems on noiseless channels," *Inf. Contr.*, vol. 514, pp. 285–298, 1969.

[20] N. Cott, “Characterization and Design of Optimal Prefix Codes,” Ph.D. dissertation, Dept. Comp. Sci., Stanford Univ., Stanford, CA, Jun. 1977.

[21] D. Altenkamp and K. Melhorn, “Codes: Unequal probabilies, unequal letter costs,” *J. Assoc. Comput. Mach.*, vol. 27, no. 3, pp. 412–427, Jul. 1980.

[22] J. Abrahams, “Code and parse trees for lossless source encoding,” *Commun. Inf. and Syst.*, vol. 1, no. 2, pp. 113–146, Apr. 2001.

[23] Y. N. Patt, “Variable length tree structures having minimum average search time,” *Commun. ACM*, vol. 12, no. 2, pp. 72–76, 1969.

[24] T. Berger and R. W. Yeung, “Optimum ‘1’ ended binary prefix codes,” *IEEE Trans. Inf. Theory*, vol. 36, no. 6, pp. 1435–1441, Dec. 1990.

[25] R. M. Capocelli, A. D. Santis, and G. Persiano, “Binary prefix codes ending in a “1”,” *IEEE Trans. Inf. Theory*, vol. 40, no. 4, pp. 1296–1302, Jul. 1994.

[26] J.-Y. Lin, Y. Liu, and K.-C. Yi, “Balance of 0, 1 bits for Huffman and reversible variable-length coding,” *IEEE Trans. Commun.*, vol. 52, no. 3, pp. 359–361, Mar. 2004.

[27] R. Sedgewick and P. Flajolet*, An Introduction to the Analysis of Algorithms*. Boston, MA: Addison-Wesley Longman, 1996.

[28] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.* vol. 27, pp. 379–423, 623–656, Jul., Oct. 1948 [Online]. Available: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf