

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

Mordecai Golin* and Jian Li**

¹ Dept of Computer Science & Engineering, HKUST, Hong Kong, China

² Dept of Computer Science & Engineering, Fudan University, Shanghai, China

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 optimally solving it, there are many good heuristics that all provide additive errors to optimal. The additive error in these algorithms usually depends linearly upon the size of the largest encoding letter.

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.

1 Introduction

Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_t\}$ be an *encoding alphabet*. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a non-empty word. A *Code* over Σ is a collection of words $C = \{w_1, \dots, w_n\}$. Code C is *prefix-free* if for all $i \neq j$ w_i is not a prefix of w_j . See Figure 1.

Let $cost(w)$ be the *length* or number of characters in w . Given a set of associated probabilities $p_1, p_2, \dots, p_n \geq 0$, $\sum_i p_i = 1$, 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 [13] ($O(tn)$ -time if the p_i are sorted in non-decreasing order).

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 $w = \sigma_{i_1}\sigma_{i_2}\dots\sigma_{i_l}$ 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

* golin@cs.ust.hk. Work partially supported by HK RGC Competitive Research Grant 613105.

** lijian83@fudan.edu.cn. Work done while visiting HKUST.

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

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

Fig. 1. Two codes for $\Sigma = \{a, b\}$. Code $\{aaa, aab, ab, b\}$ is prefix-free. Code $\{aaa, aab, ab, aaba\}$ 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 $cost(b) = 3$.

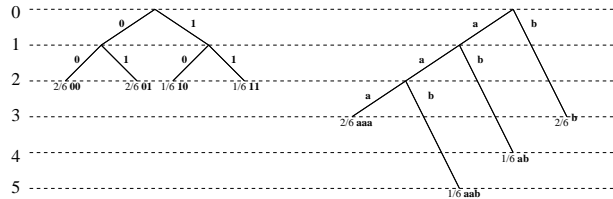


Fig. 2. Two min-cost prefix-free codes for probabilities $2/6, 2/6, 1/6, 1/6$ and their tree representations. The code on the left is optimal for $c_1 = c_2 = 1$ while the code on the right, the prefix-free code from Figure 1, is optimal for $c_1 = 1, c_2 = 3$.

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*, which, as will briefly be described in Section 3 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 [16, 18]¹ that $\frac{1}{c}H(p_1, \dots, p_n) \leq OPT$ where $H(p_1, \dots, p_n) = -\sum_{i=1}^n p_i \log p_i$ 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 $\log_2 x$.

The known efficient algorithms create a code T that satisfies

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

where $C(T)$ is the cost of code T , $\mathcal{C} = (c_1, c_2, \dots, c_t)$ 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 $\frac{1}{c}H(p_1, \dots, p_n) \leq OPT$, 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 $c_t = \max(\mathcal{C})$, the largest letter cost. They therefore can not be used for infinite \mathcal{C} . In this paper we present a new algorithmic variation (all 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:

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 [16] showed how to extend it to infinite t in cases where a positive root of $1 = \sum_{i=1}^{\infty} 2^{-cc_i}$ exists.

- (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 8) If \mathcal{C} is infinite but $d_j = |\{m \mid j \leq c_m < j + 1\}|$ is bounded, then we can still give a bound of type (1).
- (Theorem 4) If \mathcal{C} is infinite but d_i is unbounded then we can not provide a bound of type (1) but, as long as $\sum_{i=1}^{\infty} c_m 2^{-cc_m} < \infty$, we can show that

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

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

We now provide some more history and motivation. The unequal letter cost coding problem was originally motivated by coding problems in which different characters have different transmission times or storage costs see e.g., [15]. Blachman [4], Marcus [17], and (much later) Gilbert [9] 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 [15]. Later exact algorithms based on dynamic programming were given by Golin and Rote [10] for arbitrary t and a slightly more efficient one by Bradford et. al. [5] for $t = 2$. These algorithms run in $n^{\theta(c_t)}$ 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 [12] provide a polynomial time approximation scheme (PTAS). Their algorithm is mainly theoretical and not useful in practice. Finally, in contrast to the non-alphabetic case, alphabetic coding has a polynomial-time algorithm $O(tn^3)$ time algorithm [14].

Karp's result was followed by many efficient algorithms [16, 8, 7, 18, 2]. As mentioned above, $\frac{1}{c} H(p_1, \dots, p_n) \leq OPT$; almost all of these algorithms produce codes of cost at most $C(T) \leq \frac{1}{c} H(p_1, \dots, p_n) + f(\mathcal{C})$ 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 [18] which provides a bound of

$$cC(T) - H(p_1, \dots, p_n) \leq (1 - p_1 - p_n) + cc_t \quad (3)$$

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

In this paper we are only interested in the general coding problem and not the *alphabetic* one² 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 $p_1 \geq p_2 \geq p_3 \geq \dots$ and $c_1 \leq c_2 \leq c_3 \leq \dots$. These assumptions are the starting point that will permit us to derive better bounds. Furthermore, for

² *Alphabetic coding* is the same problem with the additional constraint that the code-words must be chosen in increasing alphabetic order (with respect to the words to be encoded).

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 [1, Section 2.7], which contains a section on the problem.

Due to lack of space in this extended abstract, most of the proofs have been omitted. They are available in the full paper [11].

2 Notations and definitions

There is a very standard correspondence between prefix-free codes over alphabet Σ and $|\Sigma|$ -ary trees in which the m^{th} child of node v is labelled 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. Because of this correspondence we will speak about codes and trees interchangeably.

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 exist 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 $p_1, p_2, \dots, p_n \geq 0$ is a probability distribution then the redundancy of T relative to the p_i is $R(T; p_1, \dots, p_n) = C(T) - \frac{1}{c} H(p_1, \dots, p_n)$. We will also define the normalized redundancy to be $\text{NR}(T; p_1, \dots, p_n) = cR = cC(T) - H(p_1, \dots, p_n)$. If the p_i and T are understood, we will write $R(T)$ ($\text{NR}(T)$) or even R (NR).

3 Examples of Unequal-Cost Letters

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 $\mathcal{L}^* = \bigcup_{i=0}^{\infty} \mathcal{L}^i$.

We start with cost vector $\mathcal{C} = \{1, 2, 3, \dots\}$ i.e., $\forall m > 0, c_m = m$. An early use of this problem was [19]. 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 m^{th} pointer corresponds to using character σ_m . The time that it takes to *find*

the m^{th} pointer is proportional to the location of the pointer in the list. Thus (after normalizing time units) $c_m = m$.

We now consider a generalization of the problem of **1**-ended codes. The problem of finding min-cost prefix-free code with the additional restriction that all codewords end with a 1 was studied in [3, 6] with the motivation of designing self-synchronizing codes. One can model this problem as follows. Let \mathcal{L} be a language. In our problem, $\mathcal{L} = \{w \in \{0, 1\}^* \mid \text{the last letter in } w \text{ is a } 1\}$. 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} .

Now suppose further that \mathcal{L} has the special property that $\mathcal{L} = \mathcal{Q}^*$ where \mathcal{Q} is itself a prefix-free language. Then 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 $\text{cost}(w) = \sum_{i=1}^r \text{cost}(q_i)$. 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 length of σ_q is $\text{cost}(q)$) and then solving the minimal cost coding problem for $\Sigma_{\mathcal{Q}}$ with those associated costs. For the example of **1**-ended codes we set $\mathcal{Q} = \{1, 01, 001, 0001, \dots\}$ and thus have $\mathcal{C} = \{1, 2, 3, \dots\}$ i.e., an infinite alphabet with $c_m = m$ for all $m \geq 1$.

Now consider generalizing the problem as follows. 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 $\mathcal{L} = \Sigma^* \Sigma' = \{w \in \Sigma^* \mid \text{the last letter in } w \text{ is in } \Sigma'\}$. Now note that $\mathcal{L} = D^*$ where $D = (\Sigma - \Sigma')^* \Sigma'$ 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 Σ_D and \mathcal{C}_D . The important observation is that $d_j = |\{d \in \Sigma_D \mid \text{cost}(d) = j\}|$, the number of letters in Σ_D of length j , satisfies a linear recurrence relation. Bounding redundancies for these types of \mathcal{C} will be discussed in Section 6, Case 4.

As an illustration, consider $\Sigma = \{1, 2, 3\}$ with $\mathcal{C} = (1, 1, 2)$ and $\Sigma' = \{1\}$; our problem is find minimal cost prefix-free codes in which all words end with a 1. $\mathcal{L} = \{1, 2, 3\}^* \{1\} = D^*$, where $D = \{2, 3\}^* \{1\}$. The number of characters in Σ_D with length j is $d_1 = 1, d_2 = 1, d_3 = 2, d_4 = 3, d_5 = 5$, and, in general, $d_{i+2} = d_{i+1} + d_i$, so $d_i = F_i$, the Fibonacci numbers.

4 The algorithm

All of the provably efficient heuristics for the problem, e.g., [16, 8, 7, 18, 2], use the same basic approach, which itself is a generalization of Shannon's original binary splitting algorithm [20]. 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 $p_{l_m}, p_{l_m+1}, \dots, p_{r_m}$ 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 $p_{l_m}, p_{l_m+1}, \dots, p_{r_m}$ 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.

Here we use a generalization of the version introduced in [18]. The algorithm first preprocesses the input and calculates all $P_k = p_1 + p_2 + \dots + p_k$ ($P_0 = 0$) and $s_k = p_1 + p_2 + \dots + p_{k-1} + \frac{p_k}{2}$. 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 . It 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 it “falls” into the “range” $[L_m, R_m)$.

If the interval p_k 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 [18] is that p_k is assigned to bin m if $s_k = p_1 + p_2 + \dots + p_k/2$ 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, \dots, p_r in which every code word starts with prefix U . To build the entire code we call $CODE(1, n, \lambda)$, where λ is the empty string. Figure 3 gives pseudocode.

```

CODE(l, r, U);
{Constructs codewords  $U_l, U_{l+1}, \dots, U_r$  for  $p_l, p_{l+1}, \dots, p_r$ .
  $U$  is previously constructed common prefix of  $U_l, U_{l+1}, \dots, U_r$ .}
If  $l = r$ 
    then codeword  $U_l$  is set to be  $U$ .
else {Distribute  $p_i$ s into initial bins  $I_m^*$ }
     $L = P_{l-1}; R = P_r; w = R - L$ 
     $\forall m$ , let  $L_m = L + w \sum_{i=1}^{m-1} 2^{-cc_i}$  and  $R_m = L_m + w2^{-cc_m}$ .
        set  $I_m^* = \{k \mid L_m \leq s_k < R_m\}$ 
    {Shift the bins to become final  $I_m$ . Afterwards,
     all bins  $> M$  are empty, all bins  $\leq M$  non-empty and  $\forall m \leq M, I_m = \{l_m, \dots, r_m\}$ }
    {shift left so there are no empty “middle” bins.}
     $M = 0; k = l;$ 
    while  $k \leq r$  do
         $M = M + 1;$ 
         $l_M = k; r_M = \max \{k\} \cup \{i > k \mid i \in I_M^*\};$ 
         $k = r_M + 1;$ 
    {If all  $p_i$ 's are in first bin, 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
         $CODE(l_m, r_m, U\sigma_m);$ 

```

Fig. 3. 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 actually constructed.

Assume that we currently have a prefix of U assigned to p_l, \dots, p_r . Let v be 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 ranges³ 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 $I_m^*(v) = \{k \mid L_m \leq s_k < R_m\}$.

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. See figure 4.

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 $M(v) = 2$).

Finally, after creating the 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 $CODE(l_m, r_m, U\sigma_m)$



Fig. 4. The splitting procedure creates the bins I_m^* on the left. The shifting procedure then creates the I_m on the right.

Indeed, we can show the algorithm can be implemented with a running time bounded by $n + \sum_{v \in N_T} \log n M(v) = O(n \log n)$ with no dependence upon t . The high level idea is we do not actually construct the bins $I_m^*(v)$ first. We can more efficiently construct the $I_m(v)$ using a binary search each time. The implementation details are omitted here and can be found in the full paper [11].

For comparison, we point out the algorithm in [18] 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 t^{th} bin (if they were not already there).

We will now see that our modified shifting procedure not only permits a finite algorithm for infinite encoding alphabets, but, in conjunction with the added assumption that the p_i are sorted in non-decreasing order, also often provides a provably better approximation for *finite* encoding alphabets.

5 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 $w(v) = \sum_{m=1}^t w_m^*(v) = \sum_{m=1}^t w_m(v) = \sum_{k=l}^r p_k$. We first need some Lemmas from [18].

³ In the description, t is permitted to be finite or infinite.

Lemma 1 [18] Let T be a code tree and N_T be the set of all internal nodes of T . Then

1. The cost $C(T)$ of the code tree T is $C(T) = \sum_{v \in N_T} \sum_{m=1}^t c_m \cdot w_m(v)$.
2. The entropy $H(p_1, p_2, \dots, p_n)$ is $H(p_1, p_2, \dots, p_n) = \sum_{v \in N_T} w(v) \cdot H\left(\frac{w_1(v)}{w(v)}, \frac{w_2(v)}{w(v)}, \dots\right)$.

Lemma 1 permits expressing the normalized redundancy of T as

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

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

Note that $NR(T) = \sum_{v \in N_T} w(v) \left(\sum_{m=1}^t E(v, m) \right)$. For convenience we will also define $E^*(v, m) = \frac{w_m^*(v)}{w(v)} \left(\log 2^{cc_m} + \log \frac{w_m^*(v)}{w(v)} \right)$ and $NR^*(T) = \sum_{v \in N_T} w(v) \left(\sum_{m=1}^t E^*(v, m) \right)$. The analysis proceeds by bounding the values of $NR^*(T)$ and $NR(T) - NR^*(T)$.

Lemma 2 [18]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^*, \dots 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 $f = \max I_m^*$. $E^*(v, m) \leq \frac{p_e + p_f}{w(v)}$. Furthermore, if $m = 1$ then $E^*(v, m) \leq \frac{p_f}{w(v)}$, while if $m = t$, then $E^*(v, m) \leq \frac{p_e}{w(v)}$.

Corollary 3 If the p_i are sorted in nonincreasing order then in case (c) of Lemma 2, if $m = 1$, $E^*(v, m) \leq \frac{p_f}{w(v)}$, while if $m > 1$, then $E^*(v, m) \leq \frac{2p_e}{w(v)}$.

The following lemma bounds the gap between NR and NR^* .

Lemma 4 $NR - NR^* \leq c(c_2 - c_1) \sum_{i \in A} p_i$ where $A = \{i \mid i \text{ is right shifted at some step}\}$.

Lemma 5 $NR^* \leq 2(1 - p_1) + \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v) E^*(v, m)$.

Combining this Lemma with Lemma 4 gives

Corollary 6 $NR \leq 2(1 - p_1) + c(c_2 - c_1) \sum_{i \in A} p_i + \sum_{v \in N_T} \sum_{\substack{1 \leq m \leq t \\ |I_m^*(v)|=1}} w(v) E^*(v, m)$.

We will now see different bounds on the last summand in the above expression. Section 6 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 can not 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).

⁴ slightly rewritten for our notation

Theorem 1 *If $t < \infty$ then $NR \leq 2(1 - p_1) + cc_t$.*

For a tighter analysis we will need a better bound for the case $|I_m^*| = 1$. The following simple lemma is a direct result from our splitting procedure.

Lemma 7 (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 $\frac{p_i}{w(v)} \leq 2 \cdot \sum_{j=m}^{m'} \frac{1}{2^{c \cdot c_j}} \leq 4 \cdot \sum_{j=m}^{m'-1} \frac{1}{2^{c \cdot c_j}}$.**

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

We can now show our first improved bound.

Theorem 2 *If $\beta < \infty$ then $NR \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log \beta)$.*

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

Theorem 3 *If t is finite then $NR \leq 2(1 - p_1) + \max(c(c_2 - c_1), 1 + \log t)$.*

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. By evaluating β and from Theorem 2, we can get the following simple lemma.

Lemma 8 *If $d_j = O(1)$, then $NR = O(1)$. In particular, if $\forall j, d_j \leq K$ then $\beta \leq \frac{2^c K}{1 - 2^{-c}}$ so, from Theorem 2, $NR \leq 2(1 - p_1) + \max\left(c(c_2 - c_1), 1 + c + \log\left(\frac{K}{1 - 2^{-c}}\right)\right)$.*

For general infinite alphabets we are not able to derive a constant redundancy bound but we obtain the following theorem.

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

$$R \leq \epsilon \frac{1}{c} H(p_1, \dots, p_n) + f(\mathcal{C}, \epsilon) \quad (4)$$

where $f(\mathcal{C}, \epsilon)$ is some constant based only on \mathcal{C} and ϵ . Note that this is equivalent to stating that $C(T) \leq (1 + \epsilon)OPT + f(\mathcal{C}, \epsilon)$.

6 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: $\mathcal{C}_\alpha = (c_1, c_2, \dots, c_{t-1}, \alpha)$ with $\alpha \uparrow \infty$:

We assume $t \geq 3$ 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 $(NR_\alpha) R_\alpha$ be the (normalized) redundancy corresponding to \mathcal{C}_α .

For any fixed α , the old bound (3) would give that both of NR_α and R_α tend to ∞ as α increases. Compare this to Theorem 3 which gives a uniform bound of $NR_\alpha \leq 2(1 - p_1) + \max(c^{(c_t-1)}(c_2 - c_1), 1 + \log t)$ and $R_\alpha \leq \frac{NR_\alpha}{c^{(\alpha)}} \leq \frac{2(1-p_1) + \max(c^{(c_t-1)}(c_2 - c_1), 1 + \log t)}{\bar{c}}$.

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 $\alpha \rightarrow \infty$. The bound from Theorem 3 is $NR_\alpha \leq 2(1 - p_1) + \max(c^{(\alpha)}(c_2 - c_1), 1 + \log t) \leq 3 + \log 3$ independent of α . Since $c^{(\alpha)} \geq \bar{c} = 1$ we also get $R_\alpha = \frac{NR_\alpha}{c^{(\alpha)}} \leq 3 + \log 3$.

Case 2: A finite alphabet that approaches an infinite one.

Let \mathcal{C} be an infinite sequence of letter costs such that there exists a $K > 0$ satisfying for all j , $d_j = |\{i \mid j \leq c_i < j\}| \leq K$. Let c be the root of the characteristic equation $1 = \sum_{i=1}^{\infty} 2^{-cc_i}$. Let $\Sigma^{(t)} = \{\sigma_1, \dots, \sigma_t\}$ and its associated letter costs be $\mathcal{C}^{(t)} = \{c_1, \dots, c_t\}$. 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 $NR_t \leq (1 - p_1 - p_n) + c^{(t)} c_t$ which goes to ∞ as t increases. Lemma 8 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 $NR_t \leq 2(1 - p_1) + \max\left(c(c_2 - c_1), 1 + c + \log \frac{K}{1 - 2^{-c^{(2)}}}\right)$.

Example 2 Let $\mathcal{C} = (1, 2, 3, \dots)$. i.e., $c_m = m$. The old bounds (3) gives an asymptotically infinite error as $\alpha \rightarrow \infty$. 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 $c^{(2)} = 1 - \log(\sqrt{5} - 1) \approx 0.694\dots$. Plugging into our equations gives $NR_t \leq 4.388$ and $R_t \leq 6.232$.

Case 3: An infinite case when $d_j = O(1)$. This permits applying Lemma 8 directly.

Example 3 Let \mathcal{C} contain d copies each of $i = 1, 2, 3, \dots$, 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 $R = NR \leq 2(1 - p_1) + 2$. 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 $NR \leq 2(1 - p_1) + \left(1 + \log\left(\frac{K}{1 - 2^{-c}}\right)\right) \leq 3 + \log(d+1)$, $R \leq 1 + \frac{3}{\log(d+1)}$.

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 $h(x) = \sum_{j=x}^{\infty} d_j j \alpha^j = O\left(\sum_{j=x}^{\infty} j^{d-1} j \left(\frac{\alpha}{\gamma}\right)^j\right) = O\left(x^d \left(\frac{\alpha}{\gamma}\right)^x\right)$, implying $h^{-1}(\epsilon) = \log_{\gamma/\alpha} 1/\epsilon + O(\log \log 1/\epsilon)$ where we define $h^{-1}(\epsilon) = \max\{x \mid h(x) \leq \epsilon, h(x-1) > \epsilon\}$. Working through the proof of Theorem 4 we find that when the c_m are all integral, for all m' , $g(m') = \sum_{m \geq m'} c_m \alpha^{c_m} \leq \sum_{j \geq c_{m'}} j d_j \alpha^j = h(c_{m'})$. Recall that $m_\epsilon = \max\{m \mid c_m \leq N_\epsilon\}$. Then $g(m_\epsilon) \leq h(N_\epsilon)$. Since $g(m_\epsilon) \leq \epsilon/6$, $N_\epsilon \leq h^{-1}(\epsilon/6) = \log_{\gamma/\alpha} 1/\epsilon + O(\log \log 1/\epsilon)$ and thus our algorithm creates a code T satisfying

$$C(T) - OPT \leq \epsilon OPT + \log_{\gamma/\alpha} 1/\epsilon + O(\log \log 1/\epsilon). \quad (5)$$

Example 4 Consider the case where $d_j = F_j$, the j^{th} Fibonacci number, $F_1 = 1$, $F_2 = 1$, $F_3 = 2, \dots$. (5) gives a bound on the cost of the redundancy of our code with $\frac{\gamma}{\alpha} = \frac{2}{(1+\sqrt{5})(\sqrt{2}-1)} \approx 1.492 \dots$

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 \mathcal{L} . i.e., all words which contain exactly as many 0's as 1's. Note that $\mathcal{L} = \mathcal{D}^*$ where \mathcal{D} is the set of all non-empty balanced words w such that no prefix of w is balanced. Let $d_j = |\{w \in \mathcal{D} \mid \text{cost}(w) = j\}|$. From standard generating function, we can show $d_j = 0$ 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 i^{th} Catalan number. Plugging d_j into the characteristic function and using some generating function technique, we show that $c = 1$. But on the other hand, $\sum_{m=1}^{\infty} c_m x^{c_m} = \sum_{j=1}^{\infty} j d_j x^j = 2 \sum_{j=1}^{\infty} \binom{2(j-1)}{j-1} (x^2)^j = \frac{x^2}{\sqrt{1-4x^2}}$ does not converge when $x = 1/2$. Thus, we can not 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.

7 Open Questions

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 is whether it is possible to improve Theorem 4 for some general \mathcal{C} to get a purely additive error rather than a multiplicative one combined with an additive one?

Finally, it would be interesting to devise an analysis that would work for cases where the root exists but $\sum_{m=1}^{\infty} c_m 2^{-cc_m} = \infty$, such as case 5.

References

1. Julia Abrahams. Code and parse trees for lossless source encoding. *Communications in Information and Systems*, 1(2):113–146, April 2001.
2. Doris Altenkamp and Kurt Melhorn. Codes: Unequal probabilities, unequal letter costs. *Journal of the Association for Computing Machinery*, 27(3):412–427, July 1980.
3. Toby Berger and Raymond W. Yeung. Optimum '1' ended binary prefix codes. *IEEE Transactions on Information Theory*, 36(6):1435–1441, 1990.
4. N. M. Blachman. Minimum cost coding of information. *IRE Transactions on Information Theory*, PGIT-3:139–149, 1954.
5. 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. *Journal of Algorithms*, 42:277–303, 2002.
6. Renato M. Capocelli, Alfredo De Santis, and Giuseppe Persiano. Binary prefix codes ending in a "1". *IEEE Transactions on Information Theory*, 40(4):1296–1302, 1994.
7. Norbert Cott. *Characterization and Design of Optimal Prefix Codes*. PhD Thesis, Stanford University, Department of Computer Science, June 1977.
8. I. Csisz'ar. Simple proofs of some theorems on noiseless channels. *Inform. Contr.*, 514:285–298, 1969.
9. E. N. Gilbert. Coding with digits of unequal costs. *IEEE Trans. Inform. Theory*, 41:596–600, 1995.
10. M. Golin and G. Rote. A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs. *IEEE Transactions on Information Theory*, 44(5):1770–1781, 1998.
11. Mordecai Golin and Li Jian. More efficient algorithms and analyses for unequal letter cost prefix-free coding. available at <http://arxiv.org/abs/0705.0253>, 2007.
12. Mordecai J. Golin, Claire Kenyon, and Neal E. Young. Huffman coding with unequal letter costs. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 785–791, 2002.
13. D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. IRE 40*, volume 10, pages 1098–1101, September 1952.
14. I. Itai. Optimal alphabetic trees. *SIAM J. Computing*, 5:9–18, 1976.
15. Richard Karp. Minimum-redundancy coding for the discrete noiseless channel. *IRE Transactions on Information Theory*, IT-7:27–39, January 1961.
16. R. M. Krause. Channels which transmit letters of unequal duration. *Inform. Contr.*, 5:13–24, 1962.
17. R.S. Marcus. *Discrete Noiseless Coding*. M.S. Thesis, MIT E.E. Dept, 1957.
18. K. Mehlhorn. An efficient algorithm for constructing nearly optimal prefix codes. *IEEE Trans. Inform. Theory*, 26:513–517, September 1980.
19. Yale N. Patt. Variable length tree structures having minimum average search time. *Commun. ACM*, 12(2):72–76, 1969.
20. Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.