Reserved-Length Prefix Coding

Michael B. Baer
vLnks
Mountain View, CA 94041-2803, USA
Email: calbear@ieee.org

***Abstract*— Huffman coding finds an optimal prefix code for a given probability mass function. Consider situations in which one wishes to find an optimal code with the restriction that all codewords have lengths that lie in a user-specified set of lengths (or, equivalently, no codewords have lengths that lie in a complementary set). This paper introduces a polynomial-time dynamic programming algorithm that finds optimal codes for this reserved-length prefix coding problem. This has applications to quickly encoding and decoding lossless codes. In addition, one modification of the approach solves any quasiarithmetic prefix coding problem, while another finds optimal codes restricted to the set of codes with g codeword lengths for user-specified g (e.g., $g = 2$). For small enough g, a sublinear-time constant-space approach is even more efficient.**

I. Introduction

A source emits symbols drawn from the alphabet $\mathcal{X} = \{1, 2, \ldots, n\}$. Symbol i has probability p_i, thus defining probability mass function vector $\boldsymbol{p}$. We assume without loss of generality that $p_i > 0$ for every $i \in \mathcal{X}$, and that $p_i \leq p_j$ for every $i > j$ $(i, j \in \mathcal{X})$. The source symbols are coded into binary codewords. The codeword c_i corresponding to symbol i has length l_i, thus defining length vector $\boldsymbol{l}$.

It is well known that Huffman coding [1] yields a prefix code minimizing expected value given the natural coding constraints: the integer constraint, $l_i \in \mathbb{Z}_+$, and the Kraft (McMillan) inequality [2]:

$$\kappa(\boldsymbol{l}) \triangleq \sum_{i \in \mathcal{X}} 2^{-l_i} \leq 1. \tag{1}$$

An exchange argument (e.g., [3, pp. 124-125]) easily shows that an optimal code exists which has monotonic nondecreasing lengths. Thus we can assume without loss of generality that such minimum-redundancy codes have $l_i \geq l_j$ for every $i > j$ $(i, j \in \mathcal{X})$.

There has been much work on solving this problem with other costs (objectives) and/or additional constraints [4]. One especially useful constraint [5], [6] is that of length-limited coding, in which

$$l_i \in \{1, 2, \ldots, l_{\max}\} \forall i$$

for some $l_{\max}$. A constraint that has received less attention is the *reserved-length* constraint:

$$l_i \in \Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{|\Lambda|}\} \forall i$$

for $\lambda_i \in \mathbb{Z}_+ \forall i$. In this case, instead of restricting the range of codeword lengths to an interval as in length-limited coding, it is restricted to an arbitrary set of lengths. (As demonstrated in the next section, there is no loss of generality in assuming this set to be finite). The problem is well-formed if and only if $\lambda_{|\Lambda|} \geq \log_2 n$. In such problems, if the cumulative distribution function is $s_j \triangleq \sum_{i=1}^{j} p_i$, ω_m is the number of codewords of length λ_m (so that $\sum_m \omega_m = n$), and $\Omega_m \triangleq \sum_{k=1}^{m} \omega_k$, then the expected codeword length is

$$\sum_{i \in \mathcal{X}} p_i l_i = \lambda_{|\Lambda|} - \sum_{m=1}^{|\Lambda|-1} (\lambda_{m+1} - \lambda_m) s_{\Omega_m}.$$

Since the Kraft inequality then becomes

$$\sum_{m=1}^{|\Lambda|-1} (2^{\lambda_{|\Lambda|} - \lambda_m} - 1)\omega_m \leq 2^{\lambda_{|\Lambda|}} - n$$

and $-s_j$ is a convex function on j, this is a convex optimization problem if not restricted to the integers. Therefore, approximation techniques based on convex optimization would be useful, particular for large n. With this integer restriction, however, an exact solution must be obtained in a different manner.

This problem was the focus of research at BellCore around 1989 to 1990, but, due to the lack of a solution, never published [7]. A practical application is that of fast data decompression. Perhaps the greatest bottleneck in fast Huffman decoding is the determination of codeword length from input bits, which can be done using a lookup table, a linear search, or a decision tree, depending on the complexity of the code involved [5]. The average time taken by a linear search or an optimal decision tree increases with the number of possible codeword lengths, so limiting the number of possible codeword lengths can make decoding faster; if the resulting increase in expected codeword length is small or zero, this can be an effective way of trading off compression and speed, with no compression on one end of the spectrum and optimal compression on the other end.

Consider the optimal prefix code for random variable Z drawn from the Zipf distribution with $n = 2^{12}$, that is,

$$\mathbb{P}[Z = i] = \frac{1}{i \sum_{j=1}^{n} j^{-1}}$$

which is approximately equal to the distribution of the n most common words in the English language [8, p. 89]. This code has codewords of 13 different lengths, with an average length of about 8.78 bits. If one were to restrict this code to only allow codewords of lengths in $\{5, 9, 14\}$, the resulting optimal restricted code would have an average length of about 9.27

bits. Although suboptimal, this restricted code would decode more quickly than the optimal unrestricted code.

An $O(n^4)$-time $O(n^3)$-space dynamic programming approach, introduced shortly, finds optimal reserved-length binary prefix codes. Variants of this algorithm solve a related length constraint and any case of the quasiarithmetic coding problem introduced by Campbell [9], extending the result of [10].

II. Preliminaries

Many prefix coding problems — most notably binary Huffman coding and binary length-limited "Huffman" coding — must return an optimal code in which the Kraft inequality (1) is satisfied with equality, that is, for which $\kappa(\boldsymbol{l}) = 1$. For nonbinary problems, although the corresponding inequality is not always satisfied with equality, a simple modification to the problem changes this, causing the inequality to always be equal for optimal codes [1], [11]. This is not the case for the reserved-length problem. For example, if $n = 3$ and the allowed lengths are 1 and 3, then the optimal code must have lengths 1, 3, and 3, resulting in a code for which $\kappa(\boldsymbol{l}) = 0.75$. Moreover, it is not clear how to determine $\kappa(\boldsymbol{l})$ for an optimal code other than to calculate the optimal code itself. The Huffman coding and most common length-limited approaches rely on $\kappa(\boldsymbol{l}) = 1$, so these methods cannot be used to find an optimal code here.

The Kraft inequality is often explained in terms of a *coding tree*. A binary coding tree is a rooted binary tree in which the leaves represent items to be coded. Along the path to a leaf, if the jth edge goes to the leftmost child, the jth bit of the codeword is a 0; otherwise, it is a 1. For a finite code tree, the Kraft inequality is an equality if and only if every node has 0 or 2 children. This assumption needs to be relaxed for finding an optimal reserved-length prefix code.

One approach that does not require $\kappa(\boldsymbol{l}) = 1$ is dynamic programming. Many prefix coding solutions use dynamic programming techniques [4], e.g., finding optimal codes for which all codewords end with a '1' bit [12], a situation in which, necessarily, a finite code cannot have $\kappa(\boldsymbol{l}) = 1$. For the current problem, the dynamic programming algorithm should find, for increasing tree heights, a set of (partial) candidate trees from which to choose, and it should terminate when the longest allowed (feasible) length is encountered. We thus take a similar approach to that in [12], which is itself a variant of the approach used in [13] to find optimal codes given different output symbol costs. First, however, we have to find the aforementioned longest feasible length, since we didn't specify that Λ, the set of allowed lengths, needed to be upper-bounded by any function of n or even finite.

Theorem 1: Any codeword l_i of an optimal reserved-length code either satisfies $l_i \leq n-2$ or $l_i = \lambda_\infty$, where λ_∞ is the smallest element of Λ that satisfies $\lambda_\infty > n-2$.

Proof: We first show that no partial Kraft sum of x items

$$\kappa(\boldsymbol{l}, x) \triangleq \sum_{i=1}^{x} 2^{-l_i}$$

can be in the open interval $(1-2^{-x}, 1)$, and, furthermore, if the longest codeword is of length $l_x > x-1$, the sum cannot be in $(1-2^{-x+1}+2^{-l_x}, 1)$. This is shown by induction on codeword lengths of nondecreasing order. Clearly

$$\kappa(\boldsymbol{l}, 2) = 2^{-l_1} + 2^{-l_2} \notin (3/4, 1)$$

satisfies this. Suppose the Kraft sum for $x-1$ items cannot fall in $(1-2^{-x+1}, 1)$, that is, for any code for which $\kappa(\boldsymbol{l}, x-1) < 1$, $\kappa(\boldsymbol{l}, x-1) \leq 1-2^{-x+1}$. Since the xth term is a power of two, the partial sum of a code is no greater than $1-2^{-x+1}+2^{-x} = 1-2^{-x}$ for $\kappa(\boldsymbol{l}, x) < 1$. Moreover, if $l_x \geq x$, the partial sum is less than or equal to $1-2^{-x+1}+2^{-l_x}$.

Now suppose there is an optimal code for n items which includes codeword lengths l_μ and l_ν, where $n-2 < l_\mu < l_\nu$. Assume without loss of generality that l_μ and l_ν are the longest codeword lengths and $l_\nu = l_n$ (i.e., l_ν is the longest codeword length). Note that $l_\nu \geq n$ and the Kraft sum cannot equal 1 for any code in which the longest codeword has length equal to or exceeding n; it is well known that the deepest full tree is a terminated unary tree, one with depth $n-1$. Thus $\kappa(\boldsymbol{l}) < 1-2^{-n}$. Consider a code with lengths $l'_i = l_i$ for $i < n$ and $l'_n = l_\mu$. We show that a prefix code exists with these lengths and thus achieves greater compression, rendering $\boldsymbol{l}$ suboptimal. If $l_\nu = l_\mu + 1$, then

$$\kappa(\boldsymbol{l}') = \kappa(\boldsymbol{l}) - 2^{-l_\mu - 1} + 2^{-l_\mu} \leq 1 - 2^{-n} + 2^{-l_\mu - 1} \leq 1$$

since $n \leq l_\mu + 1$. Otherwise, $l_n = l_\nu \geq n$, and

$$\kappa(\boldsymbol{l}') = \kappa(\boldsymbol{l}) - 2^{-l_\nu} + 2^{-l_\mu} \leq 1 - 2^{-n+1} + 2^{-l_\mu} \leq 1$$

since $n - 1 \leq l_\mu$. ∎

III. Algorithm

Since an optimal tree exists with has monotonic nondecreasing lengths, ω_m, the number of leaves on each "allowed level" λ_m, fully specifies this tree. For such an optimal tree, $l_i \leq \lambda_m$ have a partial Kraft sum

$$\kappa_{\lambda_m}(\boldsymbol{l}) \triangleq \kappa(\boldsymbol{l}, \upsilon_m) = \sum_{k=1}^{m} \omega_k 2^{-\lambda_k}$$

for υ_m such that $l_{\upsilon_m} \leq \lambda_m$ and either $l_{1+\upsilon_m} \leq \lambda_m$ or $\upsilon_m = n$. This Kraft sum is a multiple of $2^{-\lambda_m}$, so there exists an η_m such that $\kappa(\boldsymbol{l}, \upsilon_m) = 1 - \eta_m 2^{-\lambda_m}$, and this η_m is the number of internal nodes on level λ_m of any coding tree corresponding to the codeword lengths.

In an optimal coding tree, if Δ_m is defined to be $\lambda_{m+1} - \lambda_m$, then, for any $\upsilon_m < n$,

$$\underbrace{\eta_m 2^{\Delta_m} - (2^{\Delta_m} - 2)}_{\substack{\text{internal nodes next minus} \\ \text{single-node expansion factor}}} \quad \leq \quad \underbrace{n - \upsilon_m}_{\text{leaves under } m} \quad . \tag{2}$$

This can be seen by observing that, if a code violates this, we can produce a code with the same lengths for l_1 through l_{υ_m} and assign $l_{\upsilon_m+1} = \lambda_m$ and $l_i = \lambda_{m+1}$ for $i > \upsilon_m + 1$, and the new code would have no length exceeding that of the original code; in fact, l_{υ_m+1} is strictly shorter, so the original

code could not be optimal. For $\lambda_{m+1} = \lambda_m + 1$, this condition is identical to

$$2\eta_m \leq n - \upsilon_m \tag{3}$$

which is a looser necessary condition for optimality. For similar reasons, no optimal tree will have a partial tree with $\upsilon_m = n - 1$ for any m, since using an internal node on level λ_m for the final item results in an improved tree.

Such properties can be used to construct a dynamic programming algorithm. In describing this algorithm, we use the following notation; mnemonics are in boldface and previously defined values are included for clarity:

λ_m :	mth allowed length
λ', λ'' :	λ_m, λ_{m-1} for current m
s_i :	cumulative distribution $\sum_{j=1}^{i} p_j$
υ_m :	**u**sed u**p** leave**s** at or above level λ_m
$\upsilon_{\min}, \upsilon_{\max}$:	bounds on feasible υ_m given υ_{m-1}
η :	nod**e**s in**t**erna**l** at level λ''
$\Upsilon[m, \upsilon, \eta]$:	leaves above level λ_m
$L[m, \upsilon, \eta]$:	partial cost $\sum_{i=1}^{\upsilon_m} p_i l_i + \lambda_m \sum_{i=1+\upsilon_m}^{n} p_i$
$L_{\min}$:	expected length of best (done) tree (so far)
$m_{\min}$:	m s.t. longest codeword of best tree is λ_m
$\eta_{\min}$:	$\eta_{m_{\min}}$ for best tree
$\chi_{\min}$:	**c**odewords of lengt**h** $\lambda_{\min}$ **i**n best tree

The idea for the algorithm is to calculate the optimal $L[m, \upsilon_m, \eta_m]$ given feasible values of partial trees ($\upsilon_m < n - 1$) and to separately keep track of the best finished tree ($\upsilon_m = n$) as the algorithm progresses. The trees grow by level (λ_m for increasing m determining the outer loop). The algorithm calculates, in its inner loops for each m, all feasible values of υ_m (which are in $[0, n-2]$ for partial trees) and η_m (which are in $[0, \lfloor n/2 \rfloor]$ for partial trees due to (3); if $\eta_m > n/2$, at least one node on a lower level could be shortened to length λ_m, resulting in a strictly improved code). Thus there are $O(n^2)$ values per level, and we can try all feasible combinations, growing from the prior level with $\upsilon_{\min}$ being $\max(\upsilon_{m-1}, 2(\upsilon_{m-1} + \eta_{m-1}2^{\lambda' - \lambda''}) - n)$ and $\upsilon_{\max}$ being $\min(\upsilon_{m-1} + \eta_{m-1}2^{\lambda' - \lambda''}, n-2)$ due to the previous bounds on η_m and the fact that υ_m must be nondecreasing. We calculate L for all combinations of partial trees (saving the best combinations at a given point) and finished trees (saving $L_{\min}$, $m_{\min}$, $\eta_{\min}$, and $\chi_{\min}$ for only the best finished tree encountered up to this point). Only strictly improved finished trees get saved over older ones, so that the returned code is of minimal maximum length among optimal codes. In cases where $|\Lambda|$ is much smaller than n, additional constraints can be made, based on (2), but we do not discuss them here.

After finishing level $\lambda_{|\Lambda|}$, the optimal tree (code) is rebuilt via backtracking. Assuming arithmetic operations are constant-time, complexity of the dynamic programming algorithm is $O(|\Lambda| n^3)$-time and $O(|\Lambda| n^2)$-space. Because $|\Lambda| < n$ without loss of generality, if we assume arithmetic operations are constant time, time complexity should be $O(n^4)$ and space complexity $O(n^3)$.

Algorithm 1 Dynamic programming method for reserved-length prefix coding (with initialization of costs to ∞ and subsequent backtracking omitted for space)

Require: Cumulative distribution function $s(\cdot)$ corresponding to $\boldsymbol{p}$ of size n (not to be confused with η), Λ for which (without loss of generality) $\lambda_{|\Lambda|-1} \leq n - 2$

$L[0, 0, 1] \leftarrow 0$ {Trivial tree cost}
$\lambda'' \leftarrow 0$ {Previous level}
for $m \leftarrow 1, |\Lambda|$ {Level by level} **do**
  $\lambda' \leftarrow \lambda_m$ {Current level}
  for all $(\upsilon, \eta) \in [0, n-2] \times [0, \lfloor n/2 \rfloor]$ **do**
    if $L[m-1, \upsilon, \eta] < \infty$ **then**
      $\eta' \leftarrow \eta 2^{\lambda' - \lambda''}$ {Total nodes on new level λ_m}
      $L' \leftarrow L[m-1, \upsilon, \eta] + (\lambda' - \lambda'')(1 - s_\upsilon)$
      if $m < |\Lambda|$ {Build partial trees} **then**
        $\upsilon_{\min} \leftarrow \max(\upsilon, 2(\upsilon + \eta') - n)$
        $\upsilon_{\max} \leftarrow \min(\upsilon + \eta', n - 2)$
        for $\upsilon' \leftarrow \upsilon_{\min}, \upsilon_{\max}$ {Potential $\upsilon_m < n$} **do**
          if $L[m, \upsilon', \eta' - \upsilon' + \upsilon] > L'$ **then**
            $L[m, \upsilon', \eta' - \upsilon' + \upsilon] \leftarrow L'$ {Partial cost}
            $\Upsilon[m, \upsilon', \eta' - \upsilon' + \upsilon] \leftarrow \upsilon$
          end if
        end for
      end if
    end if
    if $n \leq \upsilon + \eta'$ **then**
      if $L' < L_{\min}$ {If best finished tree} **then**
        $L_{\min} \leftarrow L'$ {save cost, etc.}
        $(m_{\min}, \eta_{\min}, \chi_{\min}) \leftarrow (m, \eta' - n + \upsilon, \upsilon)$
      end if
    end if
  end for
  $\lambda'' \leftarrow \lambda'$ {Current level now previous level}
end for

A simple example of this algorithm at work is in finding an optimal code for Benford's law [14] with the restriction that all codeword lengths must be powers of two. In this case, p_i is $\log_{10}(i+1) - \log_{10}(i)$ for i from 1 to $n = 9$, and $\Lambda = \{1, 2, 4, 8\}$ is a sufficient range of lengths to allow, due to Theorem 1. The calculated values for each feasible partial $L[m, \upsilon, \eta]$ are shown in Table I.

On the first level, λ_1, average length is identical to the level number, and, if, for example, $\lambda_1 = 1$, the nodes at the level can include zero ($(\upsilon, \eta) = (0, 2)$), one ($(1, 1)$), or two ($(2, 0)$) terminating nodes, which are the only nontrivial entries in a two-dimensional grid for this level, as indicated by the first grid in Table I. From each nontrivial entry in the level λ_1 grid, all allowed combinations of terminating and expanding are considered until the second (level λ_2) grid is arrived at, and the algorithm proceeds similarly until all allowed levels are accounted for. Each tree with $\upsilon = n$ (all leaves accounted for) is compared with the best one so far in order to find an optimal tree. In the example, this is a tree with two codewords

$p_{\text{Benford}} \approx \{0.301, 0.176, 0.125, 0.097, 0.079, 0.067, 0.058, 0.051, 0.046\}, \quad \Lambda = (1, 2, 4, 8)$

Level $\lambda_1 = 1 \quad (m = 1)$

	$v_1 = 0$	1	2	3	4	5	6	7
$\eta_1 = 0$	∞	∞	1.000(0)	∞	∞	∞	∞	∞
1	∞	1.000(0)	∞	∞	∞	∞	∞	∞
2	**1.000 (0)**	∞	∞	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞	∞

Level $\lambda_2 = 2 \quad (m = 2)$

	$v_2 = 0$	1	2	3	4	5	6	7
$\eta_2 = 0$	∞	∞	1.523(2)	1.699(1)	2.000(0)	∞	∞	∞
1	∞	∞	1.699(1)	2.000(0)	∞	∞	∞	∞
2	∞	1.699(1)	**2.000 (0)**	∞	∞	∞	∞	∞
3	∞	2.000(0)	∞	∞	∞	∞	∞	∞
4	2.000(0)	∞	∞	∞	∞	∞	∞	∞

Level $\lambda_3 = 4 \quad (m = 3)$

	$v_3 = 0$	1	2	3	4	5	6	7
$v_3 = 0$	∞	∞	2.569(2)	2.495(3)	2.602(4)	∞	2.745(2)	2.796(3)
1	∞	∞	∞	∞	∞	2.745(2)	2.796(3)	∞
2	∞	∞	∞	∞	2.745(2)	2.796(3)	∞	∞
3	∞	∞	∞	2.745(2)	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	∞	∞	∞

TABLE I

GRIDS FOR FINDING RESERVED-LENGTH SOLUTION VIA DYNAMIC PROGRAMMING. EACH VALUE REPRESENTS OPTIMAL PARTIAL COST FOR A GIVEN η, v, AND LEVEL λ, WITH NUMBER OF LEAVES ABOVE THE GIVEN LEVEL GIVEN IN PARENTHESES. THE PARTIAL TREES USED IN THE OPTIMAL RESULT — THAT TERMINATED WITH $(m, v, \eta) = (3, 9, 2)$ HAVING $\boldsymbol{l} = \{2, 2, 4, 4, 4, 4, 4, 4, 4\}$ — ARE SHOWN IN BOLDFACE.

of length two and seven codewords of length four.

Note that a similar approach could be used for nonbinary trees, although an efficient exponentiation procedure would need to be used where simple shifting sufficed to calculate $\eta_{m-1} 2^{\lambda' - \lambda''}$ in the binary case. The aforementioned expansion bounds (anywhere we have factors of two) also need adjustment for nonbinary cases. These alternations do not worsen computational complexity.

IV. EXTENSIONS

The aforementioned method yields a prefix code minimizing expected length for a known finite probability mass function under the given constraints. However, there are many varied instances in which expected length is not the proper value to minimize [4]. Many such problems are in a certain family of generalizations of the Huffman problem introduced by Campbell in [15].

While Huffman coding minimizes $\sum_{i \in \mathcal{X}} p_i l_i$, Campbell's quasiarithmetic formulation adds a continuous (strictly) monotonic increasing *cost function* $\varphi(l) : \mathbb{R}_+ \to \mathbb{R}_+$. The value to minimize is then

$$L(\boldsymbol{p}, \boldsymbol{l}, \varphi) \triangleq \varphi^{-1} \left(\sum_{i \in \mathcal{X}} p_i \varphi(l_i) \right).$$

Convex φ have been solved for [10]. For nonconvex functions, it suffices to calculate the partial cost $L[m, v, \eta]$ from one level to another as $L \leftarrow L[i-1, v, \eta] + (\varphi(\lambda') - \varphi(\lambda''))(1 - s_v)$ instead of $L \leftarrow L[i-1, v, \eta] + (\lambda' - \lambda'')(1 - s_v)$. The exchange argument still holds, resulting in a monotonic solution, and Λ still has cardinality less than n, so the algorithm proceeds similarly for identical reasons, and thus with the same complexity. A nonbinary coding extension is similar to that used to minimize expected length.

Before discussing another extension, we should note that the aforementioned algorithm is needlessly slow for small $|\Lambda|$. For example, if $|\Lambda| = 2$ then, due to monotonicity and the Kraft inequality, the solution is independent of the probability distribution, and there should be $\lfloor (2^{\lambda_2} - n)/(2^{\lambda_2 - \lambda_1} - 1) \rfloor$ codewords of length λ_1, with the rest having length λ_2. For larger values, the problem is a convex optimization problem with a simple linear constraints, as noted in the introduction. If $|\Lambda| = 3$, one can do binary search over the differences of consecutive feasible values of λ_1, finding the minimum over feasible lengths. Given λ_1, the other values are directly calculated. This solves the problem in $O(\log n)$ time and constant space given $\{s_i\}$, a properly sorted cumulative distribution function corresponding to the sorted probability mass function. For larger values of $|\Lambda|$, standard convex optimization techniques [16] find a numerical solution not restricted to integers, i.e., one that might correspond to 2.4 codewords of length λ_1. Finding the optimal integer solution given the aforementioned numerical solution grows exponentially difficult due to the curse of dimensionality, and thus the dynamic programming approach is best for all but the smallest $|\Lambda|$.

As previously stated, one purpose for reserving lengths is to allow faster decoding by having fewer codewords. However, if this is the objective, the problem remains of how to select the codeword lengths to use. We might, for example, restrict

our solution to having two codeword lengths, but not put any restrictions on what these codeword lengths should be. Such a problem was examined analytically in [17] for n approaching infinity. Here, we consider solving the problem for fixed n.

One approach to the two-length problem would be to try all feasible combinations of codeword lengths. We then have to find a feasible set, hopefully one relatively small so as not to drastically increase the complexity of the problem.

First note that, if only one codeword length is used, then $\lambda_2 = \lambda_1 = \lceil \log_2 n \rceil$. Otherwise, we begin by observing that, for the best tree, the number of internal nodes and leaves on the first allowed level λ_1 must each be greater than 0 (or else only one codeword length could be used) and combined be no greater than $n-1$ (or else a better code exists with all codewords having one length). Thus $\lambda_1 \leq \log_2(n-1)$, or, put another way, $\lambda_1 \leq \lceil \log_2 n \rceil - 1$. At the same time, the second allowed level cannot have $2n-2$ or more combined internal nodes and leaves; otherwise an improved tree can be found by decreasing λ_2 by one, since no more than $n-1$ leaves can be on this level. Because these nodes are all descendants of all least one internal node on the first allowed level, this results in $2^{\lambda_2 - \lambda_1} < 2n-2$, which leads to $\lambda_2 - \lambda_1 \leq \lceil \log_2(n-1) \rceil \leq \lceil \log_2 n \rceil$. Combining these results, we find that $\lambda_2 \leq 2\lceil \log_2 n \rceil - 1$.

This result, while not the strictest bound possible, is sufficient for us to determine that the number of codeword length combinations one would have to try would be $O(\log^2 n)$. Thus, since $|\Lambda| = 2$ in all cases (running in constant time per case) and only constant-space data need be kept between combinations, the algorithm has only a $O(\log^2 n)$ time requirement and a $O(1)$ space requirement given $\{s_i\}$. For example, the optimal two-length code for the Benford distribution has two codewords of length two and seven codewords of length four. This also happens to be the code found above to be optimal for lengths restricted to powers of two, and has average codeword length $3.04\ldots$, very near to that of the optimal unrestricted Huffman code, which has average codeword length $2.92\ldots$.

The two-length problem's solution can be easily generalized to that of a g-length problem. For $g = |\Lambda| = 3$, we can use the aforementioned $O(\log n)$ binary search method for each feasible value, thus solving the problem in $O(\log^4 n)$ time. For an already-known pre-sorted cumulative distribution function, then, one can actually solve these problems more quickly than finding the optimal Huffman code. However, as g increases, the curse of dimensionality affects not only the convex optimization approach, but any approach that examines all feasible combinations of allowed codeword lengths. If λ_m is upper bounded by (approximately) $m \log_2 n$, then we can only say that there are no more than $g! \log_2^g n$ combinations, which is not workable except for the smallest values of g.

For large enough values of n, approximation algorithms may be called for, due to the increased granularity of the possible integer solutions. Let us briefly sketch one approach: Since codeword lengths should be around clusters of probabilities, if any, we can use Lloyd's algorithm [18] to cluster the probabilities (using "ideal lengths" $(-\log p_i)$, weighted by p_i, as the values in question). We can stop the algorithm in an early iteration or even skip it entirely, instead using equiprobable partitions. We can optimize Λ given the partition we choose. Solving yields $\lambda_k = \log_2(\omega_k / q_k)$, where $q_k \triangleq s_{\Omega_k} - s_{\Omega_{k-1}}$, the sum probability of partition k. Rounding to integer values for the lengths, we can find a corresponding optimal or approximated code. Details of such an approximation technique are beyond the scope of this discussion, although it bears noting that the expected "length" for this real-valued Λ is $\sum q_k(\log_2 \omega_k - \log_2 q_k)$, or the sum of the entropy of the partition (the bits needed to code which part of the partition the element is in) and the expected value of the logarithm of the part size (the bits needed to determine, via a fixed-length code on the part, which element of the part is in question).

Nevertheless, for small g and n the dynamic programming approach suffices, and all g'-length problems, for $g' \leq g$, can be solved with similar complexity, allowing for a selection of the desired trade-off between number of codeword lengths (speed) and expected codeword length (compression efficiency). Modifications can enact additional restrictions on codeword lengths (e.g., a limit on maximum length) in a straightforward fashion.

REFERENCES

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

[2] B. McMillan, "Two inequalities implied by unique decipherability," *IRE Trans. Inf. Theory*, vol. IT-2, no. 4, pp. 115–116, Dec. 1956.

[3] T. Cover and J. Thomas, *Elements of Information Theory*, 2nd ed. New York, NY: Wiley-Interscience, 2006.

[4] J. Abrahams, "Code and parse trees for lossless source encoding," *Communications in Information and Systems*, vol. 1, no. 2, pp. 113–146, Apr. 2001.

[5] A. Moffat and A. Turpin, "On the implementation of minimum redundancy prefix codes," *IEEE Trans. Commun.*, vol. 45, no. 10, pp. 1200–1207, Oct. 1997.

[6] I. H. Witten, A. Moffat, and T. Bell, *Managing Gigabytes*, 2nd ed. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

[7] Z. Zhang, Private communication, Feb. 2005.

[8] G. K. Zipf, "Relative frequency as a determinant of phonetic change," *Harvard Studies in Classical Philology*, vol. 40, pp. 1–95, 1929.

[9] L. L. Campbell, "Block coding and Rényi's entropy," *Int. J. Math. Stat. Sci.*, vol. 6, no. 1, pp. 41–47, June 1997.

[10] M. B. Baer, "Source coding for quasiarithmetic penalties," *IEEE Trans. Inf. Theory*, vol. IT-52, no. 10, pp. 4380–4393, Oct. 2006.

[11] ——, "D-ary bounded-length Huffman coding," in *Proc., 2007 IEEE Int. Symp. on Information Theory*, June 24–29, 2007, pp. 896–900.

[12] S.-L. Chan and M. J. Golin, "A dynamic programming algorithm for constructing optimal "1"-ended binary prefix-free codes," *IEEE Trans. Inf. Theory*, vol. IT-46, no. 4, pp. 1637–1644, July 2000.

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

[14] F. Benford, "The law of anomalous numbers," *Proc. Amer. Phil. Soc.*, vol. 78, no. 4, pp. 551–572, Mar. 1938.

[15] L. L. Campbell, "Definition of entropy by means of a coding problem," *Z. Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 6, pp. 113–118, 1966.

[16] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge Univ. Press, 2003, available at http://www.stanford.edu/~boyd/cvxbook.html.

[17] E. Figueroa and C. Houdré, "On the asymptotic redundancy of lossless block coding with two codeword lengths," *IEEE Trans. Inf. Theory*, vol. IT-51, no. 2, pp. 688–692, Feb. 2005.

[18] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer Academic Publishers, 1992.