Codes: Unequal Probabilities, Unequal Letter Costs

DORIS ALTENKAMP AND KURT MEHLHORN

University of Saarlandes, Saarbrücken, Federal Republic of Germany

ABSTRACT. The construction of alphabetic prefix codes with unequal letter costs and unequal probabilities is considered. A variant of the noiseless coding theorem is proved giving closely matching lower and upper bounds for the cost of the optimal code. An algorithm is described which constructs a nearly optimal code in linear time.

KEY WORDS AND PHRASES: codes, unequal letter costs, unequal probabilities, noiseless coding, prefix codes, approximation algorithm, search trees

CR CATEGORIES: 5.25, 5.39, 5.6

1. *Introduction*

We study the construction of prefix codes in the case of unequal probabilities and unequal letter costs. The investigation is motivated by and oriented toward the following problem. Consider the ternary search tree in Figure 1. It has three internal nodes and six leaves. The internal nodes contain the keys {3, 4, 5, 10, 12} in sorted order, and the leaves represent the open intervals between keys. The standard strategy to locate X in this tree is best described by the following recursive procedure SEARCH.

```
proc SEARCH(int $X$; node $v$)
if $v$ is a leaf
then "$X$ is not in the tree"
else begin let $K_1$, $K_2$ be the keys in node $v$;
            if $X < K_1$ then SEARCH($X$, left son of $v$)
            if $X = K_1$ then exit (found);
            if $K_2$ does not exist
            then SEARCH($X$, right son of $v$)
            else begin if $X < K_2$ then SEARCH($X$, middle son of $v$);
            if $X = K_2$ then exit (found);
            SEARCH($X$, right son of $v$)
            end
     end
end
```

Apparently the search strategy is unsymmetric. It is cheaper to follow the pointer to the first subtree than to the second subtree, and it is cheaper to locate K_1 than K_2.

We also assume that the probability of access is given for each key and each interval between keys. More precisely, suppose we have n keys $B_1, \ldots, B_n$ out of an ordered universe, with $B_1 < B_2 < \cdots < B_n$. Then β_i denotes the probability of accessing B_i, $1 \le i \le n$, and α_j denotes the probability of accessing elements X, with $B_j < X < B_{j+1}$,

A preliminary version of this paper was presented at the Fifth International Colloquium on Automata, Languages, and Programming, Udine, Italy, 1978.

Authors' address: University of Saarlandes, 66 Saarbrücken, Fachbereich 10, Federal Republic of Germany 6600.

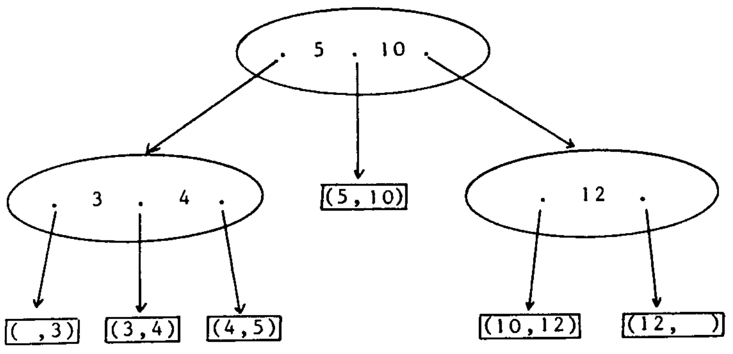


FIGURE 1

$0 \leq j \leq n$. α_0 and β_n have obvious interpretations. In our example $n = 5$, β_2 is the probability of accessing 4, and α_4 is the probability of accessing $X \in (4, 5)$. We always write the distribution of access probabilities as $\alpha_0, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n$.

Ternary trees, in general, $(t + 1)$-ary trees, correspond to prefix codes in a natural way. We are given letters $a_0, a_1, a_2, \ldots, a_{2t}$ of cost $c_0, c_1, c_2, \ldots, c_{2t}$, respectively; $c_l > 0$ for $0 \leq l \leq 2t$. Here letter a_{2l} corresponds to following the pointer to the $(l + 1)$st subtree, $0 \leq l \leq t$, and letter a_{2l+1} corresponds to a successful search terminating in the $(l + 1)$st key of a node, $0 \leq l < t$.

In our example, $t = 2$. The code word corresponding to 4, denoted W_2, is a_0a_3. The code word corresponding to (10, 12), denoted V_4, is a_4a_0.

In general, a search tree is a prefix code

$$C = \{V_0, W_1, V_1, \ldots, W_n, V_n\} \quad \text{with} \quad V_j \in \Sigma^* \quad \text{and} \quad W_i \in \Sigma^*\Sigma_{\text{end}},$$

where $\Sigma = \{a_0, a_2, a_4, \ldots, a_{2t}\}$ and $\Sigma_{\text{end}} = \{a_1, a_3, \ldots, a_{2t-1}\}$, $0 \leq j \leq n$, $1 \leq i \leq n$. Σ^* denotes the set of all words over alphabet Σ, W_i describes the search process leading to key B_i, and V_j describes the search process leading to interval (B_j, B_{j+1}).

Remark. In the binary case $t = 1$, letters a_0, a_1, a_2 have the natural interpretation $<$, $=$, and $>$. Letter a_1 (=) ends successful searches and letter a_1 is never used in unsuccessful searches. In signaling-codes applications, alphabet Σ_{end} might serve synchronizing purposes (cf. the example of an alphabetic Morse code at the end of Section 3).

Note that the use of the letters in Σ_{end} is very restricted. They can only be used at the end of code words, and they can only be used in words W_i. Furthermore, the code words must reflect the ordering of the keys, i.e.,

$$(*) \qquad V_j < W_i < V_{j'}$$

for $j < i \leq j'$, and $<$ denotes the lexicographic ordering of strings based on the ordering $a_0 < a_1 < a_2 < \cdots < a_{2t}$ of letters. The cost of a word $a_{i_1}a_{i_2}a_{i_3} \cdots a_{i_k}$ is equal to $c_{i_1} + c_{i_2} + \cdots + c_{i_k}$, i.e., the sum of the costs of the letters. The (expected) cost of code C is then defined as

$$\text{Cost}(C) = \sum_{i=1}^{n} \beta_i \,\text{Cost}(W_i) + \sum_{j=0}^{n} \alpha_j \,\text{Cost}(V_j).$$

Remark. In the binary equal cost case ($t = 1$, $c_0 = c_1 = c_2 = 1$) this definition coincides with the definitions of weighted path length used in the literature [e.g., 4, 11, 13, 15, 16].

We will address the following two problems:

(1) Given letters, their costs, and a probability distribution, find a code with nearly minimal cost.
(2) Give good a priori bounds for the cost of the optimal code.

We refer to these problems as the *alphabetic coding problems.* We will also have to consider nonalphabetic codes, i.e., codes which do not have the ordering requirement (*) on the code words and which have unlimited usage of letters. Formally, given letters $a_0, \ldots, a_s$, their costs $c_0, \ldots, c_s$, and a probability distribution $p_1, \ldots, p_n$, we want to find a prefix code $C = \{U_1, \ldots, U_n\}$ such that

$$\mathrm{Cost}(C) = \sum_{i=1}^{n} p_i \,\mathrm{Cost}(U_i)$$

is minimal.

Remark. We use the notation $p_1, \ldots, p_n$ for the probability distribution in the nonalphabetic case and $\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n$ in the alphabetic case. This should help the reader keep things apart.

We show that the cost of an optimal alphabetic code C_{opt} satisfies the following inequalities. Here $H = H(\alpha_0, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n) = -\sum\beta_i \log \beta_i - \sum\alpha_j \log \alpha_j$ is the entropy of the probability distribution, $B = \sum\beta_i$, and $c, d \in \mathbb{R}$ are such that $\sum_{k=0}^{t} 2^{-dc_{2k}} = 1$ and $\sum_{k=0}^{2t} 2^{-cc_k} = 1$. Numbers 2^{-d}, 2^{-c} are sometimes called the "roots of the characteristic equation of the letter costs" [cf. 6]. Also, log denotes logarithm base 2, and ln denotes natural logarithm.

$$H \le d \cdot \mathrm{Cost}(C_{\mathrm{opt}}) + \frac{1}{u}\, c \cdot B \max_{i\,\mathrm{odd}} c_i[1 + \ln(u \cdot v \cdot \mathrm{Cost}(C_{\mathrm{opt}})] + \frac{1}{(e \cdot u)} \tag{1}$$

for some constants u, v and $e = 2.71\ldots$;

$$\mathrm{Cost}(C_{\mathrm{opt}}) \le \frac{H}{d} + (\textstyle\sum\alpha_j)\left[\frac{1}{d} + \max\limits_{k\,\mathrm{even}} c_k\right] + (\textstyle\sum\beta_i)\left[\max\limits_{k\,\mathrm{odd}} c_k\right]. \tag{2}$$

Note that the lower and upper bound differ essentially by ln Cost(C_{opt}). Inequality (1) is proved in Corollary 1. Theorem 2 gives a better bound than Corollary 1, but the bound is harder to state. Inequality (2) is proved in Theorem 3 by the explicit construction of a code C satisfying (2). Moreover, this code can be constructed in linear time $O(t \cdot n)$ (Theorem 4).

Inequalities (1) and (2) provide us with a "noiseless coding theorem" for alphabetic coding with unequal letter costs and unequal probabilities.

The construction of prefix codes is an old problem. We close this introduction by briefly reviewing some results.

Case 1. *Equal Letter Costs* (i.e., $c_i = 1$ for all i, $0 \le i \le s$). In the *nonalphabetic* case an algorithm for the construction of an optimal code dates back to Huffmann [10]; it can be implemented to run in time $O(n \log n)$ [19]. The noiseless coding theorem (due to Shannon [18]) gives bounds for the cost of the optimal code, namely,

$$\frac{1}{\log(s+1)} H(p_1, \ldots, p_n) \le \mathrm{Cost}(C) \le \frac{1}{\log(s+1)} [H(p_1, \ldots, p_n) + 1],$$

where $H(p_1, \ldots, p_n) = -\sum p_i \log p_i$ is the entropy of the distribution.

The binary alphabetic case was solved by Gilbert and Moore [8], Knuth [13], and Hu and Tucker [9]. The time complexity of their algorithm is $O(n^2)$ and $O(n \log n)$, respectively. Cost is usually called weighted path length in this context. Bounds were proved by Bayer [4] and Mehlhorn [16], namely,

$$H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n) \le \mathrm{Cost}(C_{\mathrm{opt}}) + (\log e) - 1 + \log \mathrm{Cost}(C_{\mathrm{opt}}),$$
$$\mathrm{Cost}(C_{\mathrm{opt}}) \le H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n) + 1 + \textstyle\sum\alpha_j.$$

Various approximation algorithms exist which construct codes in linear time in the binary case. The cost of these codes lie within the above bounds [4, 7, 15, 16].

Case 2. *Equal Probabilities* (i.e., $p_i = 1/n$ for $1 \le i \le n$.) The problem was solved by Perl, Garey, and Even [17]. The time complexity of their algorithm is $O(\min(t^2 n, tn\log n))$.

The alphabetic case is identical to the nonalphabetic case, and no a priori bounds exist for the cost of an optimal code.

Case 3. *Unequal Probabilities, Unequal Letter Costs.* This case was treated by Karp [12]. He reduced the problem to integer programming and thus provides us with an algorithm of exponential time complexity. No better algorithm is known at present. However, it is also not known whether the corresponding recognition problem (is there a code of cost $\le m$) is NP-complete. A priori bounds were proved by Krause [14], Csiszar [5], and Cot [6].

The alphabetic case was treated by Itai [11]. He describes a clever dynamic programming approach which constructs an optimal alphabetic code in time $O(t^2 \cdot n^3)$. No a priori bounds are known.

2. *The Lower Bound*

In this section we want to prove a lower bound on the cost of every prefix code. We first treat the nonalphabetic case and then extend the results to the alphabetic case.

2.1 The Nonalphabetic Case

2.1.1 *Preliminary Considerations.* Consider the binary case first. There are two letters of cost c_1 and c_2, respectively. In the first node of the code tree we split the set of given probabilities into two parts of probability p and $1 - p$, respectively (Figure 2). The local information gain per unit cost is then

$$G(p) = \frac{H(p, 1-p)}{c_1 \cdot p + c_2(1-p)},$$

where $H(p, q) = -p \log p - q \log q$. This is equivalent to

$$G(p) = \frac{-p \log p - (1-p)\log(1-p)}{(-p \cdot \log 2^{-cc_1} - (1-p) \log 2^{-cc_2}) \cdot (1/c)} \qquad \text{for all} \quad c \ne 0.$$

The following fact shows that $G(p)$ is maximal for $p = 2^{-cc_1}$, $1 - p = 2^{-cc_2}$, where c is chosen such that $2^{-cc_1} + 2^{-cc_2} = 1$. Hence $G(p) \le c$ for all p, and $G(2^{-cc_1}) = c$.

Fact (cf., e.g., Ash [2]). *Let $x_i, y_i \ge 0$ for $1 \le i \le n$, $\sum x_i = 1 = \sum y_i$. Then*

$$-\sum x_i \log x_i \le -\sum x_i \log y_i.$$

This shows that the maximal local information gain per unit cost is c. Hence every code for probabilities $p_1, \dots, p_n$ should have cost at least $(1/c) \cdot H(p_1, \dots, p_n)$. This is made precise in the next section.

The plausibility argument also suggests an approximation algorithm: Try to split the given set of probabilities into two parts of probability p and $1 - p$, respectively, so as to make $|p - 2^{-cc_1}|$ as small as possible. We discuss this approach in Section 3.

2.1.2 *The Lower Bound in the Nonalphabetic Case*

Theorem 1. *Let $p_1, \dots, p_n$ be a probability distribution and let $C = \{U_1, \dots, U_n\}$ be a prefix code over code alphabet $\{a_0, \dots, a_s\}$. Let $c_i > 0$ be the cost of a_i, $0 \le i \le s$. Let c be such that $\sum_{i=0}^{s} 2^{-cc_i} = 1$.*

(*a*) [14] *$Cost(C) \ge H(p_1, \dots, p_n)/c$, where $H(p_1, \dots, p_n) = -\sum p_i \log p_i$ is the entropy of the frequency distribution.*

(*b*) *Let $h \in \mathbb{R}$, $h \ge 0$, and*

$$L_h = \{i;\ c\,Cost(U_i) \le \log p_i - h\}.$$

Then $\sum_{i \in L_h} p_i \le 2^{-h}$.

Remark. Inequality (a) reads in its full form

$$\sum_{i=1}^{n} p_i[c\mathrm{Cost}(U_i)] \ge \sum_{i=1}^{n} p_i[-\log p_i].$$

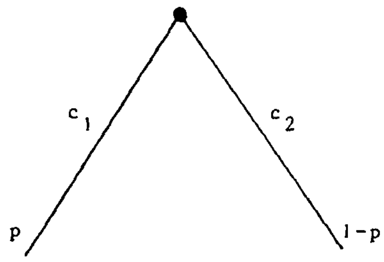


FIGURE 2

It is an extension of the noiseless coding theorem to arbitrary letter costs. Part (b) shows that this inequality is almost satisfied termwise by the expressions in square brackets. More precisely, the fraction of probabilities which violates the termwise inequality by more than h is less then 2^{-h}.

PROOF. (a) Let $U_i = a_{i_1} a_{i_2} \cdots a_{i_{l_i}}$. Define

$$q_i := \prod_{k=1}^{l_i} 2^{-cc_{i_k}}, \qquad 1 \le i \le n,$$
$$Q := \sum_{i=1}^{n} q_i.$$

Then $Q \le 1$ by a simple induction argument on max l_i. The prefix property is needed here. Furthermore,

$$\log q_i = -c \cdot \sum_{k=1}^{l_i} c_{i_k} = -c\,\mathrm{Cost}(U_i),$$

and hence, by the fact above,

$$\begin{aligned} H(p_1, \ldots, p_n) &= -\sum p_i \log p_i \\ &\le -\sum p_i \log(q_i/Q) \\ &= c\,\mathrm{Cost}(C) + \log Q \\ &\le c \cdot \mathrm{Cost}(C). \end{aligned}$$

(b) Let $h \ge 0$ and

$$L_h = \{i;\ c\,\mathrm{Cost}(U_i) \le -\log p_i - h\}.$$

Then

$$\begin{aligned} 1 \ge Q &= \sum_{i=1}^{n} 2^{-c\,\mathrm{Cost}(U_i)} \\ &\ge \sum_{i \in L_h} 2^{-c\,\mathrm{Cost}(U_i)} \\ &\ge \sum_{i \in L_h} 2^{\log p_i + h} = 2^h \cdot \sum_{i \in L_h} p_i. \end{aligned}$$ □

2.2 THE ALPHABETIC CASE. Every alphabetic code $C = \{V_0, W_1, \ldots, W_n, V_n\}$ is a nonalphabetic code, and hence Theorem 1 applies. It shows that

$$\mathrm{Cost}(C) \ge \frac{1}{c} \cdot H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n),$$

where $\sum_{k=0}^{2t} 2^{-cc_k} = 1$. In this section we improve upon this lower bound and essentially show that for every *alphabetic* code C,

$$\mathrm{Cost}(C) \ge \frac{1}{d} \cdot \left[H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n) - \frac{c}{u} \cdot \max_{i\,\mathrm{odd}} c_i \cdot \ln H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)\right],$$

where $\sum_{k=0}^{t} 2^{-dc_{2k}} = 1$ and u is some constant. Note that only the letters in Σ, and not those in Σ_{end}, are used to define d, and hence the new bound is much better for large H.

Example. Consider ternary trees with $c_0 = c_1 = c_2 = c_3 = c_4 = 1$. Then $c = \log 5$ and $d = \log 3$.

The alphabetic case differs from the nonalphabetic case in two respects:

(1) The letters in Σ_{end} can only be used at the end of code words W_i and not at all in words V_j.
(2) The lexicographic ordering of code words must reflect the underlying ordering of the keys.

We will use only restriction (1) to improve upon the lower bound. There seems to be no way to incorporate this (combinatorial) restriction into the proof of Theorem 1. Rather, we turn the combinatorial restriction into a constraint on costs by artificially increasing the cost of letters in Σ_{end}. Then we use the fact that letters in Σ_{end} are used at most once in words W_i and not at all in words V_j in order to relate the cost of a code under the old and the new cost function. Finally, we apply Theorem 1 to the new cost function. Let $1 \leq x < \infty \cdots$ be arbitrary, let

$$\tilde{c}_i = c_i \quad \text{for } i \text{ even,}$$
$$\tilde{c}_i = x \cdot c_i \quad \text{for } i \text{ odd,}$$

and let $c(x) \in \mathbb{R}$ be such that $\sum_{k=0}^{2t} 2^{-c(x)\tilde{c}_k} = 1$.

Remark. In the new cost function $\tilde{c}_i$, $0 \leq i \leq 2t$, we increased the cost of letters in Σ_{end} by factor x. For $x = 1$ the new cost function is identical with the old, and hence $c(1) = c$; for $x = \infty$ the cost of letters in Σ_{end} is infinite, and hence $c(\infty) = d$.

Let $C = \{V_0, W_1, V_1, \ldots, W_n, V_n\}$ be an *alphabetic code* for probability distribution $(\alpha_0, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n)$. In particular, $V_j \in \Sigma^*$ and $W_i \in \Sigma^*\Sigma_{\text{end}}$. Let $\widetilde{\text{Cost}}(C)$ be the cost of C with respect to $\tilde{c}_0, \tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_{2t}$, and let Cost$(C)$ be the cost of C with respect to $c_0, c_1, \ldots, c_{2t}$.

LEMMA 1. *$\widetilde{Cost}(C) \leq Cost(C) + (x-1) \cdot B \cdot \max_{i\,odd} c_i$ for every x, $1 \leq x \leq \infty$, $B = \sum_{i=1}^{n} \beta_i$.*

PROOF. For $W_i \in \Sigma^*\Sigma_{\text{end}}$ let

$$W_i = W_i' \cdot a_{j_i}, \qquad a_{j_i} \in \Sigma_{\text{end}}.$$

Then

$$\begin{aligned} \widetilde{\text{Cost}}(W_i) &= \widetilde{\text{Cost}}(W_i') + \tilde{c}_{j_i} \\ &= \text{Cost}(W_i') + x \cdot c_{j_i} \\ &= \text{Cost}(W_i) + (x-1)c_{j_i}. \end{aligned}$$

Hence

$$\begin{aligned} \widetilde{\text{Cost}}(C) &= \sum \beta_i \widetilde{\text{Cost}}(W_i) + \sum \alpha_j \widetilde{\text{Cost}}(V_j) \\ &\leq \text{Cost}(C) + (x-1) \cdot B \max_{i\,\text{odd}} c_i. \end{aligned}$$ □

We next use Theorem 1 for the costs $\tilde{c}_i$, $0 \leq i \leq 2t$.

THEOREM 2. *Let $c(x)$ be such that $\sum_{k=0}^{2t} 2^{-c(x)\tilde{c}_k} = 1$. Then*

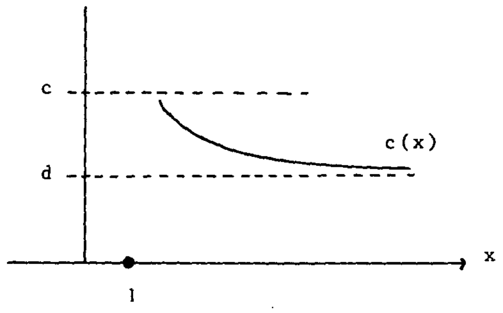
$$Cost(C) \geq \max\{H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)/c(x) - (x-1) \cdot B \cdot \max_{i\,odd} c_i;\ 1 \leq x \leq \infty\}.$$

PROOF. By Theorem 1,

$$\widetilde{\text{Cost}}(C) \geq \frac{H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)}{c(x)}.$$

Substituting into Lemma 1 yields the result. □

We were unable to find a closed form expression for the maximal value of the right-hand side in Theorem 2. An approximate value can be found as follows. Recall that $c(1) = c$, $c(\infty) = d$, and $c(x)$ decreases for $1 \le x \le \infty$. Write $c(x) = d + \delta(x)$.



with $0 \le \delta(x) \le c - d$. We show that $\delta(x) \le v \cdot e^{-u(x-1)}$ for some constants u, v (see Lemma 2 below). Then Theorem 1 can be written as (we write H instead of $H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)$):

$$\begin{aligned} H &\le c(x) \cdot \mathrm{Cost}(C) + (x-1) \cdot c(x) \cdot B \max_{i\,\mathrm{odd}} c_i \\ &\le d \cdot \mathrm{Cost}(C) + \delta(x) \cdot \mathrm{Cost}(C) + (x-1) \cdot c \cdot B \cdot \max_{i\,\mathrm{odd}} c_i \\ &\le d \cdot \mathrm{Cost}(C) + v \cdot e^{-u(x-1)} \cdot \mathrm{Cost}(C) + (x-1) \cdot c \cdot B \cdot \max_{i\,\mathrm{odd}} c_i. \end{aligned}$$

This inequality is true for all x, $1 \le x \le \infty$.

The right-hand side is minimal (differential calculus) for

$$(x-1) = \frac{(\ln[u \cdot v\ \mathrm{Cost}(C)/c \cdot B \cdot \max_{i\,\mathrm{odd}} c_i])}{u}.$$

Hence

$$H \le d \cdot \mathrm{Cost}(C) + \frac{c \cdot B}{u} \max_{i\,\mathrm{odd}} c_i \left[1 + \ln \left(\frac{u \cdot v \cdot \mathrm{Cost}(C)}{c \cdot B \cdot \max_{i\,\mathrm{odd}} c_i} \right) \right].$$

Finally, using $y \ln(1/y) \le 1/e$ for all $y > 0$ (in particular, $y = (cB \max c_i)/u$), we obtain

COROLLARY 1. *Let C be an alphabetic code for distribution $\alpha_0, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n$ with respect to costs $c_0, c_1, \ldots, c_{2t}$. Let c, d be such that*

$$\sum_{k=0}^{2t} 2^{-cc_k} = 1, \qquad \sum_{k=0}^{t} 2^{-dc_{2k}} = 1.$$

Let $B = \sum \beta_i$. Then there are constants u, v (depending on $c_0, c_1, \ldots, c_{2t}$ but not on $Cost(C)$ and $\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n$) such that

$$H(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n) \le d \cdot Cost(C) + \frac{cB}{u} \cdot \max_{i\,odd} c_i[1 + ln(u \cdot v Cost(C))] + \frac{1}{e \cdot u}.$$

PROOF. By the preceding argument. □

Corollary 1 shows that the lower bound for the alphabetic code is essentially the lower bound ($d \cdot \mathrm{Cost}(C)$) for the nonalphabetic code where only the letters of even index are used, plus a small correction of order ($c \cdot B \cdot \max_{i\,\mathrm{odd}} c_i \ln \mathrm{Cost}(C)$) which reflects the restricted usage of the letters in Σ_{end}.

A special case of Theorem 2 and Corollary 1 was proved by Bayer [4]. He considered the binary alphabetic case with equal letter costs, i.e., $t = 1$ and $c_0 = c_1 = c_2 = 1$.

It remains to prove Lemma 2. We will show the existence of constants u, v but not derive a bound for them. This is justified since we recommend always using Theorem 2 and computing the maximal value of the right-hand side by numerical methods. Corollary 1 is only given in order to indicate the order of the bound in Theorem 2.

LEMMA 2. *Let $\delta(x)$ be defined as above. Then*

$$\delta(x) \le v \cdot e^{-u(x-1)}$$

for some constants u, v.

PROOF. $\delta(x) \le v \cdot e^{-u(x-1)}$ is equivalent to $(x-1) \le -\ln(\delta(x)/v)/u$. $\delta(x)$ is defined by

$$\sum_{k=0}^{t} 2^{-(d+\delta(x))c_{2k}} + \sum_{k=1}^{t} 2^{-(d+\delta(x))\cdot x \cdot c_{2k-1}} = 1.$$

Consider the left-hand side as a function $f(x, \delta)$ of two arguments x and δ; i.e., replace $\delta(x)$ by δ in the left-hand side. For fixed δ this function is decreasing in x. Also, $f(x, \delta(x)) = 1$. Suppose we know that $f(z, \delta(x)) \le 1$ for some z. Then $x \le z$, since $z < x$ implies $f(x, \delta(x)) < f(z, \delta(x)) \le 1$, a contradiction. It therefore suffices to show that there are constants u, v such that for all x,

$$\sum_{k=0}^{t} 2^{-(d+\delta(x))c_{2k}} + \sum_{k=1}^{t} 2^{-(d+\delta(x))zc_{2k-1}} \le 1, \tag{3}$$

where $z := 1 - \ln(\delta(x)/v)/u$. Replacing c_i, $0 \le i \le 2t$, by $c_{\min} = \min\{c_i; 0 \le i \le 2t\} > 0$ in the left-hand side of (3) only increases the left-hand side. It therefore suffices to show that

$$2^{-\delta(x)c_{\min}} \cdot \sum_{k=0}^{t} 2^{-dc_{2k}} + t2^{-dzc_{\min}} \le 1 \tag{4}$$

for some constants u, v. Using $\sum_{k=0}^{t} 2^{-dc_{2k}} = 1$, the left-hand side of (4) is of the form

$$g(y) := b_1^{-y} + b_2(y/v)^{b_3},$$

with $b_1 = 2^{c_{\min}} > 1$, $b_2 = t2^{-dc_{\min}} > 0$, $b_3 = (dc_{\min} \ln 2)/u > 0$, and $y = \delta(x)$. Hence $0 \le y \le c - d$. Choose u such that $b_3 = 1$. Then

$$g(y) = b_1^{-y} + b_2(y/v).$$

It remains to show that we can choose v such that $g(y) \le 1$ for $0 \le y \le c - d$. Note that $g(0) = 1$ and that

$$\begin{aligned} g'(y) &= (-\ln b_1)b_1^{-y} + \frac{b_2}{v} \\ &\le (-\ln b_1)b_1^{-(c-d)} + \frac{b_2}{v} \qquad \text{since} \quad 0 \le y \le c-d \\ &\le 0 \end{aligned}$$

for sufficiently large v. Hence $g(y) \le 1$ for $0 \le y \le d$. This shows the existence of u and v. □

3. *The Upper Bound*

In this section we describe an algorithm for constructing alphabetic codes and derive a bound on the cost of the code constructed. The algorithm is a generalization of the one in [8, 16].

The code is constructed top-down by repeated splitting of the ordered set $\{\alpha_0, \beta_1, \alpha_1, \ldots, \alpha_{n-1}, \beta_n, \alpha_n\}$ of probabilities. In each step we try to split the set as described in 2.1.1. Let d be such that

$$\sum_{k=0}^{t} 2^{-dc_{2k}} = 1,$$

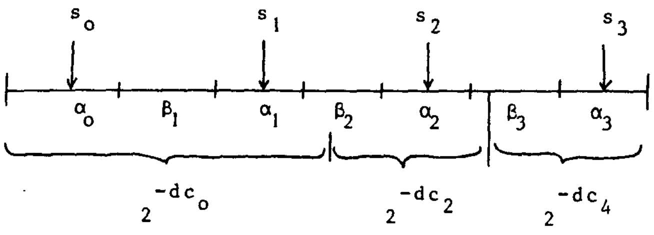


FIGURE 3

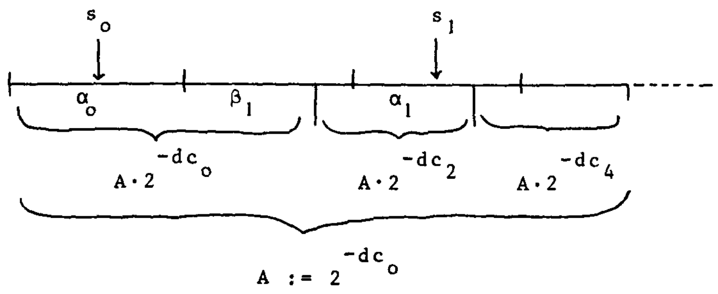


FIGURE 4

and let

$$s_{-1} = -\infty, \qquad s_{n+1} = \infty, \qquad s_0 = \alpha_0/2,$$
$$s_i = \alpha_0 + \beta_1 + \cdots + \beta_i + \alpha_i/2 \qquad \text{for} \quad 1 \le i \le n.$$

s_{-1} and s_{n+1} are defined as "stoppers."

Example. Let $c_0 = 1$, $c_1 = 3$, $c_2 = 2$, $c_3 = 1$, $c_4 = 2$. Then $d = 1$. Let $\alpha_0 = \alpha_i = \beta_i = \frac{1}{7}$ for $1 \le i \le 3$. Then $s_i = (4i + 1)/14$ for $0 \le i \le 3$. We draw the distribution $(\alpha_0, \beta_1, \alpha_1, \ldots, \alpha_{n-1}, \beta_n, \alpha_n)$ as a partition of the unit interval and split the unit interval in the ratio 2^{-dc_0}: $2^{-dc_2}:2^{-dc_4}$.

From Figure 3, it appears reasonable to assign letter a_0 to α_0, β_1, α_1, to assign letter a_2 to α_2, letter a_4 to α_3, letter a_1 to β_2, and letter a_3 to β_3. In other words, we set $W_2 = a_1$, $V_2 = a_2$, $W_3 = a_3$, and $V_3 = a_4$, and let V_0, W_1, V_1 start with a_0. Next we have to work on the subproblem $\{\alpha_0, \beta_1, \alpha_1\}$. We split the interval $[0, 2^{-dc_0}]$ in the same way and obtain Figure 4. This suggests the use of letter $a_0(a_1, a_2)$ as the second letter of the code words assigned to $\alpha_0(\beta_1, \alpha_1)$. Note that we used letter a_2 for α_1, since more than half of probability α_1 falls into the interval of length $A \cdot 2^{-dc_2}$.

In general, the construction process can be described as a recursive procedure CODE with parameters

(1)
- l, r — We work on the subproblem $\alpha_l, \beta_{l+1}, \ldots, \beta_r, \alpha_r$; $l \le r$;
- L, R — $L, R \in \mathbb{R}$, $L \le s_l \le s_r \le R$;
- U — $U \in \Sigma^* = \{a_0, a_2, \ldots, a_{2t}\}^*$. U is a common prefix of code words $V_l, W_{l+1}, V_{l+1}, \ldots, W_r, V_r$; and

(2) $R - L = 2^{-d\cdot \mathrm{Cost}(U)}$.

Initially $l = 0$, $r = n$, $L = 0$, $R = 1$, and $U = \epsilon$ where ϵ is the empty word. Consider now any call of the procedure CODE with parameters l, r, L, R, U satisfying the invariants (1) and (2) stated in their definition.

Case 1. $l = r$. Then we define $V_r = U$ and return.

Case 2. $l < r$. We split the interval (L, R) in the ratio $2^{-dc_0}:2^{-dc_1}:\ldots:2^{-dc_{2t}}$. The ith

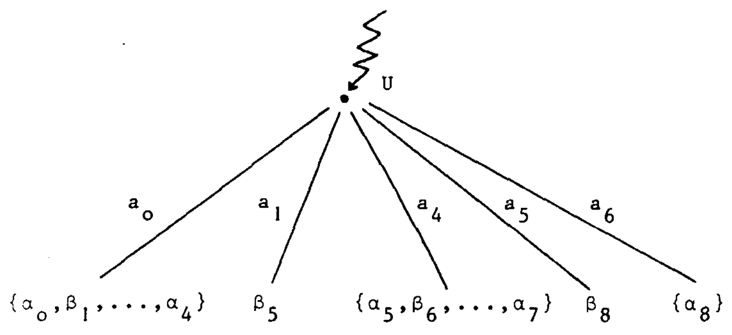

U
a_0
a_1
a_4
a_5
a_6
$\{\alpha_0, \beta_1, \ldots, \alpha_4\}$
β_5
$\{\alpha_5, \beta_6, \ldots, \alpha_7\}$
β_8
$\{\alpha_8\}$

FIGURE 5

subinterval, $0 \le i \le t$, has boundaries $L_i = L + (R - L) \cdot \sum_{k=0}^{i-1} 2^{-dc_{2k}}$ and $R_i = L_i + (R - L) \cdot 2^{-dc_{2i}}$. We then determine for each subinterval the set of s_k's which lie in that subinterval, say $S_{h-1} \le L_i < S_h$ and $S_j \le R_i < S_{j+1}$ for the ith interval. If $h \le j$, i.e., some s_k's actually lie in the ith subinterval, then we call procedure CODE recursively with parameters $l = h$, $r = j$, $L = L_i$, $R = R_i$, $U = Ua_{2i}$. Furthermore, if in addition $j + 1 \le r$, then we assign code word Ua_{2i+1} to β_{j+1}; i.e., we set $W_{j+1} = Ua_{2i+1}$.

Example. Suppose $t = 3$ and $L_0 \le s_0 \le \cdots \le s_4 < L_1 < L_2 < s_5 \le \cdots \le s_7 \le L_3 < s_8 \le R_3$. Then the recursive calls are CODE(0, 4, L_0, L_1, Ua_0), CODE(5, 7, L_2, L_3, Ua_4), and CODE(8, 8, L_3, R_3, Ua_6). Furthermore, we set $W_5 = Ua_1$ and $W_8 = Ua_5$. A pictorial representation is given in Figure 5.

In the remainder of this section we derive an upper bound on the cost of the code constructed by procedure CODE. It is obvious that the properties stated in the definitions of l, r, L, R, U are invariants of the recursive procedure; i.e., they hold for all values of the actual parameters.

Consider the code word $W_i = Ua_{k_i}$ constructed for β_i; $U \in \Sigma^*$ and $a_{k_i} \in \Sigma_{\text{end}}$. The word W_i was constructed by the procedure CODE with actual parameters l, r, L, R, U, where $l < i \le r$. Hence

$$\beta_i \le \alpha_l/2 + \beta_{l+1} + \alpha_{l+1} + \cdots + \beta_q + \alpha_r/2,$$

since β_i appears in that sum, and thus

$$\begin{aligned}\beta_i &\le s_r - s_l\\ &\le R - L = 2^{-d\,\text{Cost}(U)}\end{aligned}$$

by invariants (1) and (2) of procedure CODE. Hence

$$\begin{aligned}\text{Cost}(W_i) &\le \text{Cost}(U) + \max_{K\text{ odd}} c_K\\ &\le \frac{1}{d}[-\log \beta_i] + \max_{K\text{ odd}} c_K.\end{aligned}$$

Consider next code word V_j. Word V_j was constructed by procedure CODE with actual parameters $(j, j, \ , \ , V_j)$. CODE with actual parameters $(j, j, \ , \ , V_j)$ was called by CODE with actual parameters (l, r, L, R, U), with $l < r$, $l \le j \le r$, and $V_j = Ua_{k_j}$ for some $a_{k_j} \in \Sigma$. Hence

$$\begin{aligned}\alpha_j/2 &\le \alpha_l/2 + \beta_{l+1} + \alpha_{l+1} + \cdots + \beta_r + \alpha_r/2\\ &= s_r - s_l \le R - L = 2^{-d\,\text{Cost}(U)}\end{aligned}$$

by the same reasoning as above. Hence

$$\text{Cost}(V_j) \le \frac{1}{d}[-\log \alpha_j + 1] + \max_{k\text{ even}} c_k.$$

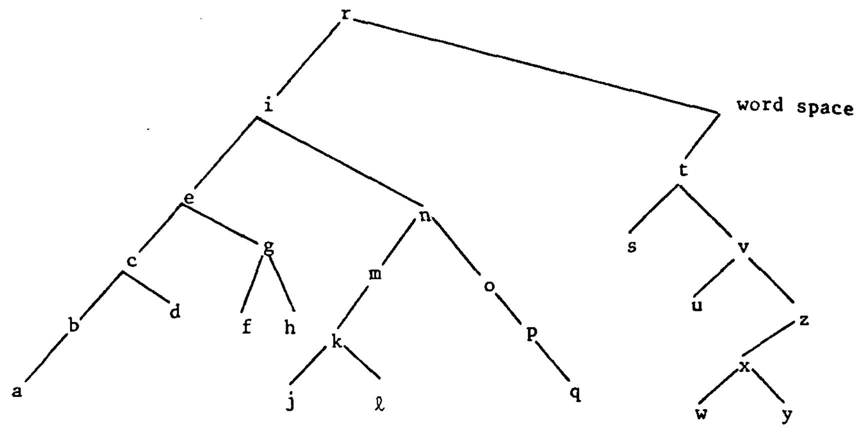


FIGURE 6

We summarize:

THEOREM 3. *Let $(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)$ be a probability distribution, $\beta_i \geq 0$, $\alpha_j \geq 0$, $\sum\beta_i + \sum\alpha_j = 1$. Let $a_0, a_1, \ldots, a_{2t}$ be $(2t + 1)$ symbols with costs $c_0, c_1, \ldots, c_{2t} \in \mathbb{R}_+$. Then procedure CODE constructs an* alphabetic *code with*

(*a*) $Cost(W_i) \leq [-log\ \beta_i]/d + max_{k\ odd}\ c_k$;
(*b*) $Cost(V_j) \leq [-log\ \alpha_j + 1]/d + max_{k\ even}\ c_k$;
(*c*) $Cost(C) \leq H(\alpha_0, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n)/d + (\sum\alpha_j)[1/d + max_{k\ even}\ c_k] + (\sum\beta_i)[max_{k\ odd}\ c_k]$.

PROOF. (a) and (b) are proved by the discussion above. (c) follows from (a) and (b) by multiplication with β_i and α_j, respectively, and summation. □

Example. An ordered Morse code. The Morse code is over a three letter alphabet: dot (cost 1), dash (cost 2), and letter space (cost 1). We assume the ordering dot < letter space < dash; i.e., Σ = {dot, dash} and Σ_{end} = {letter space}. Then $c_0 = 1$, $c_1 = 1$, $c_2 = 2$, $2^{-d} = 0.618$, and $d = 0.6942$. We encode the 27 English letters (including the word space) in alphabetical ordering; i.e., β_1 = probability of letter a, β_2 = probability of letter $b, \ldots,$ β_{27} = probability of word space. We refer the reader to [3] for the extract values of $\beta_1, \beta_2, \ldots, \beta_{27}$. All α_j's are zero. Then $H(\alpha_0, \beta_1, \ldots, \beta_{27}, \alpha_{27}) = 4.1$. The lower bound of Theorem 2 is

$$\text{Cost}(C) \geq \max\{4.1/c(x) - (x - 1);\ 1 \leq x \leq \infty\},$$

where $c(x)$ is such that $2^{-c(x)} + 2^{-2c(x)} + 2^{-xc(x)} = 1$. The maximal value of the right-hand side is about 3.24 with $x = 1.44$ and $c(x) = 1.19$. The upper bound of Theorem 3 is 5.85. The code actually constructed is shown in Figure 6; i.e., r is encoded by letter space, i is encoded by dot letter space, and n by dot dash letter space. The cost of this code is 4.3025. In comparison, the cost of the morse code is 4.055. The Morse code is nonalphabetic.

4. *Implementation*

In this section we describe an implementation of procedure CODE. Our implementation has running time $O(t \cdot n)$. As above, let $d \in \mathbb{R}$ be such that $\sum_{k=0}^{t} 2^{-dc_{2k}} = 1$. Furthermore, let $z_i = \sum_{k=0}^{i} 2^{-dc_{2k}}$ for $0 \leq i \leq t$. Procedure CODE has the following global structure.

```
    procedure CODE($l, r, L, R, U$);
    begin
    if $l = r$
    then $V_l \leftarrow U$
    else begin
(*)         for all $i$, $0 \leq i \leq t$ do
```

```
            begin $L_i := L + (R - L)z_{i-1}$;
                  $R_i := L + (R - L)z_i$;
(**)              let $h$ and $j$ be such that
                  $s_{h-1} \le L_i < s_h$ and $s_j \le R_j < s_{j+1}$;
                  if $h \le j$ then CODE($h, j, L_i, R_i, Ua_{2i}$);
                  if $j + 1 \le r$ then $W_{j+1} \leftarrow Ua_{2i+1}$
            end
      end
  end
```

Three problems remain to be solved:

(a) In what order do we process the different values of i in loop (*)?
(b) How do we find h and j in line (**)?
(c) What should we do if all s_i's, $l \le i \le r$, lie in the same subinterval? Note that problem (c) does not affect the analysis given in Section 3; however, it will affect running time.

Consider problem (b) first. We describe a solution for the 0th subinterval. By definition, $L_0 = L$, and hence $s_{l-1} \le L_0 \le s_l$ by assumption. Thus we only have to find j such that $s_j \le R_0 < s_{j+1}$. We find j by exponential + binary search [7]. We first compare R_0 with

$$s_{l+1}, s_{l+2}, s_{l+4}, s_{l+8} \quad \text{until} \quad s_{l+2^k} > R_0 \quad \text{or} \quad l + 2^k > r.$$

In the second case we have $s_r \le R_0$; i.e., all s_i's fall into the same interval. In the first case we have $s_{l+2^k} > R_0$ and $s_{l+2^{k-1}} \le R_0$ or $k = 0$. If k is equal to 0, then either $j = l + 1$ (if $s_l \le R_0$) or $j = l$ (if $R_0 < s_l$). If k is not equal to 0, then $l + 2^{k-1} \le j \le l + 2^k$. We determine the exact value of j by binary search on the interval $l + 2^{k-1} \cdots l + 2^k$ in time $O(k)$.

Let $n_0 = j - l + 1$; i.e., n_0 is the number of s_i's which lie in the 0th interval. Equivalently, the recursive call CODE($l, j, \ldots$) constructs $n_0 - 1$ code words W_i.

Since $j - l \ge 2^{k-1}$ where k is determined as above, it follows that j can be determined in time $\le a(1 + \log(n_0 + 1))$, where a is a suitable constant.

Next we address problem (a). Let n_i, $0 \le i \le t$, be the number of s_i's which lie in the ith interval. The obvious way to proceed is to determine $n_0, n_1, n_2, \ldots, n_t$ in that order. Note that the solution given to (b) applies to all n_i's. However, this strategy may waste a lot of time, e.g., if n_1 is large and $n_2, \ldots, n_p$ are small. Note that n_t actually does not have to be computed because it is uniquely determined once the other values are found. It would be much cheaper in this case to compute $n_1, n_2, \ldots$ in reverse order. These considerations lead to the following strategy.

Determine n_0 and n_t in parallel, and stop when any one of them is found. Say n_0 was determined first. Forget everything about n_t. Now determine n_1 and n_t in parallel....

In this way one can find $n_0, \ldots, n_t$ in time

$$a' \cdot \left(\sum_{i=0}^{t} (1 + \log(n_i + 1)) - \max_{0 \le i \le t} (1 + \log(n_i + 1)) \right)$$

for some constant a'.

It remains to treat problem (c). Suppose all but one n_i are 0, say $n_j = n$. In this case we either artificially assign the leftmost probability α_l to the 0th subinterval (if $j \ge 1$) or the rightmost probability α_r to the tth subinterval (if $j < t$). More precisely, suppose $j \ge 1$. Then we set $V_l \leftarrow Ua_0$, $W_{l+1} \leftarrow Ua_1$, and call CODE recursively with parameters $l + 1$, r, L_j, R_j, Ua_{2j}. Note that the analysis of Section 3 is still valid. By this modification we guarantee that at least one code word W_i is constructed by every call of procedure CODE. We are now ready to set up recursion equations for an upper bound T on the running time of our implementation of algorithm CODE. Let $T(n + 1, t)$ be the maximal time needed by CODE in order to construct a code for probability distribution $(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)$ and code alphabet $a_0, a_1, \ldots, a_{2t-1}, a_{2t}$ with costs $c_0, c_1, \ldots, c_{2t}$. Note that $n + 1$ is equal to the number of α_j's. Then

$$T(0, t) = 0, \qquad T(1, t) = a,$$

for some constant a.

Let $n + 1 > 1$; i.e., we have to construct a code for $(\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n)$. We first determine $n_0, n_1, \ldots, n_t$ as described above in time

$$a \cdot \left(\sum_{i=0}^{t} (1 + \log(n_i + 1) - \max_{0 \le i \le t} (1 + \log(n_i + 1)) \right).$$

Since n_i is the number of s_j's which fall in the ith subinterval, we have $n + 1 = n_0 + n_1 + \cdots + n_t$. Also, $0 \le n_i$ and $n_i \le n$ by our modification above. For every $n_i > 0$ we have to call CODE recursively; this recursive call takes time at most $T(n_i, t)$.

For the sequel, it will be convenient to modify CODE slightly. If $\max n_i > 4$, then we proceed as described above. If $\max n_i \le 4$, then we avoid recursive calls altogether. Rather, we solve each subproblem directly in time $O(t)$. This gives the following recursion equation for T (we replace $n + 1$ by n throughout):

$$T_1(n, t) = \max_{\substack{n_0 + \cdots + n_t = n \\ n_i < n \\ \max n_i > 4}} \left[\sum_{i=1}^{t} (T(n_i, t) + a(1 + \log(n_i + 1))) - \max_{0 \le i \le t} a(1 + \log(n_i + 1)) \right],$$

$$T_2(n, t) = \max_{\substack{n_0 + \cdots + n_t = n \\ n_i < n \\ \max n_i \le 4}} \left[\sum_{i=0}^{t} (\textbf{if } n_i \neq 0 \textbf{ then } a(t + 1) \textbf{ else } 0 + a(1 + \log(n_i + 1))) - \max_{0 \le i \le t} a(1 + \log(n_i + 1)) \right],$$

$$T(n, t) = \max(T_1(n, t), T_2(n, t)).$$

Here a is some constant; without loss of generality we can use the same a in all equations.

THEOREM 4. *$T(n, t) = O((t + 1) \cdot n)$.*

PROOF. We show by induction on n that

$$T(n, t) \le d(t + 1) \cdot n - e(t + 1) \cdot \log(n + 1) \tag{5}$$

for some suitable constants d and e (to be determined later).

Induction base. $n = 0$, $n = 1$, or $n = n_0 + \cdots + n_t$; $0 \le n_i < n$; $\max n_i \le 4$; and $T(n, t) = T_2(n, t)$. Then

$$T(0, t) = 0, \qquad T(1, t) = a,$$

and

$$\begin{aligned} T(n, t) &\le a(t + 1) \cdot (\text{number of } n_i\text{'s} \neq 0) + a(t + 1)(1 + \log 5) \\ &\le a(t + 1) \cdot n + a(t + 1) \log 10. \end{aligned}$$

In either case we can find for every choice of e a suitable d such that (5) is true.

Induction step. Let $n = n_0 + \cdots + n_t$, $0 \le n_i < n$, $\max n_i > 4$, and $T(n, t) = T_1(n, t)$. Then by the induction hypothesis,

$$\begin{aligned} T(n, t) \le \sum_{i=0}^{t} &[d(t + 1)n_i - e(t + 1)\log(n_i + 1) + a(1 + \log(n_i + 1)] \\ &- \max_{0 \le i \le t} a(1 + \log(n_i + 1)). \end{aligned}$$

We may assume without loss of generality that $n_0 = \max n_i$. Then

$$\begin{aligned} T(n, t) \le{} & d(t + 1) \cdot n - e(t + 1)\log(n + 1) + e(t + 1)\log(n + 1) \\ &+ \sum_{i=1}^{t} a(1 + \log(n_i + 1)) - \sum_{i=0}^{t} e(t + 1)\log(n_i + 1). \end{aligned}$$

It suffices to show that

$$e(t+1)\log(n+1) + at$$
$$\le e(t+1)\log(n_0+1) + (e(t+1)-a)\sum_{i=1}^{t}\log(n_i+1).$$

Since $\sum_{i=1}^{t}\log(n_i+1)$ is smallest when all but one n_i, $1 \le i \le t$, are zero, we have $\sum_{i=1}^{t}\log(n_i+1) \ge \log(n-n_0+1)$. Thus it suffices to show that

$$e(t+1)\log(n+1) + at$$
$$\le e(t+1)\log(n_0+1) + (e(t+1)-a)\log(n-n_0+1).$$

The derivative of the right-hand side with respect to n_0 is

$$f(n_0) := \frac{1}{\ln 2}\frac{e(t+1)n + a + (a - 2e(t+1))n_0}{(n_0+1)(n-n_0+1)}.$$

For $0 \le n_0 \le n$ the denominator is positive. The numerator is a linear function of n_0 which is positive for $n_0 = 0$. Hence there exists some real m such that $f(n_0) \ge 0$ for $0 \le n_0 \le m$ and $f(n_0) \le 0$ for $m \le n_0 \le n$. (It is conceivable that $m \ge n$.) Hence it suffices to check the inequality for the extremal values of n_0: $n_0 = n - 1$ and $n_0 = \max(n/(t+1), 5)$. For $n_0 = n - 1$ the inequality reduces to

$$e(t+1)\log(n+1) + at \le e(t+1)\log n + (e(t+1) - a),$$

or

$$e(t+1)\log\frac{n+1}{n} \le (e-a)(t+1).$$

Since $n > n_0 \ge 5$, one has only to choose e such that

$$\log\tfrac{7}{6} \le (e-a)/e.$$

Suppose now that $n_0 = \max(n/(t+1), 5)$. If $n_0 = n/(t+1) \ge 5$, and hence $n \ge 5(t+1)$, the inequality reduces to

$$e(t+1)\log\left(\frac{n+1}{n_0+1}\right) + at \le (e(t+1)-a)\log\left(\frac{t}{t+1}n+1\right).$$

Since $t \ge 1$, $(n+1)/(n_0+1) \le t+1$, and $tn/(t+1) + 1 \ge 5t + 1 = 5(t+1) - 4$, it suffices to show that

$$e(t+1)\log(t+1) + at \le (e(t+1)-a)\log(5(t+1)-4),$$

or

$$a(t + \log(5(t+1)-4)) \le e(t+1)\cdot\log\left(\frac{5(t+1)-4}{t+1}\right).$$

Since $t \ge 1$ and hence $(5(t+1)-4)/(t+1) \ge 3$, it suffices to choose e such that

$$a\left(1 + \frac{\log(5(t+1)-4)}{t+1}\right) \le e$$

for $t \ge 1$.

Finally, if $n_0 = 5 > n/(t+1)$, and hence $n < 5(t+1)$, the inequality reduces to

$$e(t+1)\log(n+1) + at \le e(t+1)\log 6 + (e(t+1)-a)\log(n-4),$$

or

$$e(t+1)\log\left(\frac{n+1}{n-4}\right) + a\log(n-4) \le e(t+1)\log 6 - at.$$

Since $5 = n_0 < n < 5(t+1)$, it suffices to show that

$$e(t+1)\log\tfrac{7}{2} + a\log 5t \le e(t+1)\log 6 - at,$$

or

$$a(t + \log 5t) \le e(t+1)\log\tfrac{12}{7}$$

for $t \ge 1$. Hence we only need to choose e sufficiently large.

In either case one only has to choose e sufficiently large in order for the induction step to carry through. Since the validity of the induction base is independent of the value of e, the theorem follows. □

Remark. If the for-loop (*) in procedure CODE is realized as **for** i **from** 0 **to** t **do**, then the recursive equation,

$$T(n,t) = \max_{\substack{n_0+\cdots+n_t=n\\ n_i<n}} \left[\sum_{i=1}^{t} T(n_i,t) + \sum_{i=1}^{t-1} a(1+\log(n_i+1))\right]$$

with solution $T(n,t) = O(tn\log n)$ arises. So the modification suggested above is essential.

Theorem 4 shows that a prefix code satisfying the inequality of Theorem 3 can be constructed in linear time $O(t\cdot n)$. Two variants of the above recursion equations for T might sometimes be useful. An application can be found in [1].

Variant A

$$T(n,t) = \max_{\substack{n_1+\cdots+n_s=n\\ 1\le n_i<n\\ 1\le s\le t}} \left[\sum_{i=0}^{s} T(n_i,t) + a(1+\log n_i)\right].$$

It has a solution $T(n,t) = O(n\log n)$ [1].

Variant B

$$T(n,t) = a \qquad \text{for} \quad n \le 4,$$

$$T(n,t) = \max_{\substack{n_0+n_1+\cdots+n_s=n\\ 1\le n_i<n\\ 1\le s\le t}} \left[\sum_{i=0}^{s} (T(n_i,t) + a(1+\log n_i)) - \max_{0\le i\le s} a(1+\log n_i)\right].$$

It has a solution $T(n,t) = O(n)$ [1].

REFERENCES

1. ALTENKAMP, D., AND MEHLHORN, K. Codes: Unequal probabilities, unequal letter costs. Tech. Rep., University des Saarlandes, Saarbrücken, Federal Republic of Germany, 1978.
2. ASH, R. *Information Theory.* Interscience, New York, 1965.
3. BAUER, F.L., AND GOOS, G. *Informatik, Heidelberger Taschenbücher.* Springer-Verlag, Berlin, 1971.
4. BAYER, P.J. Improved bounds on the costs of optimal and balanced binary search trees. Tech. Memo., Project MAC TM 69, M.I.T., Cambridge, Mass., 1975.
5. CSISZAR, I. Simple proofs of some theorems on noiseless channels. *Inf. Control 14* (1969), 285–298.
6. COT, N. Characterization and design of optimal prefix codes. Ph.D. Thesis, Stanford University, Stanford, Calif. June 1977.
7. FREDMAN, M.L. Two applications of a probabilistic search technique. Proc. 7th Ann. ACM Conf. on Theory of Computing, Albuquerque, N.M., 1975.
8. GILBERT, E.N., AND MOORE, E.F. Variable length encodings. *Bell Syst. Tech. J. 38* (1959), 933–968.
9. HU, T.C., AND TUCKER, A.C. Optimal search trees and variable length alphabetic codes. *SIAM J. Appl. Math. 21* (1971), 514–532.
10. HUFFMANN, D.A. A method for the construction of minimum-redundancy codes. *Proc. IRE 40* (1952), 1098–1101.
11. ITAI, A. Optimal alphabetic trees. *SIAM J. Comput. 5* (1976), 9–18.
12. KARP, R.M. Minimum redundancy coding for the discrete noiseless channel. *IEEE Trans. Inf. Theory IT-7* (Jan. 1961), 27–39.

13. KNUTH, D.E. Optimum binary search trees. *Acta Inform. 1* (1971), 14–25.
14. KRAUSE, R.M. Channels which transmit letters of unequal duration. *Inf. Control 5* (1962), 13–24.
15. MEHLHORN, K. *Effiziente Algorithmen*. Teubner Studienbücher Informatik, Stuttgart, 1977.
16. MEHLHORN, K. Best possible bounds on the weighted path length of optimum binary search trees. *SIAM J. Comput. 6*, 2 (1977), 235–239.
17. PERL, Y., GAREY, N.R., AND EVEN, S. Efficient generation of optimal prefix code: Equiprobable words using unequal cost letters. *J. ACM 22*, 2 (April 1975), 202–214.
18. SHANNON, C.E. A mathematical theory of communication. *Bell Syst. Tech. J. 27* (1948), 379–423, 623–656.
19. VAN LEEUWEN, J. On the construction of Huffmann trees. In *3rd International Colloquium on Automata, Languages, and Programming*, S. Michaelson and R. Milner, Eds., Edinburgh University Press, 1976, pp. 382–410.

RECEIVED JANUARY 1978; REVISED JUNE 1979; ACCEPTED JULY 1979