CODES: UNEQUAL PROBABILITIES, UNEQUAL LETTER COSTS

BY

DORIS ALTENKAMP AND KURT MEHLHORN

A 78/18 IX/1978

Fachbereich lo
Universität des
Saarlandes
BRD-66oo Saarbrücken

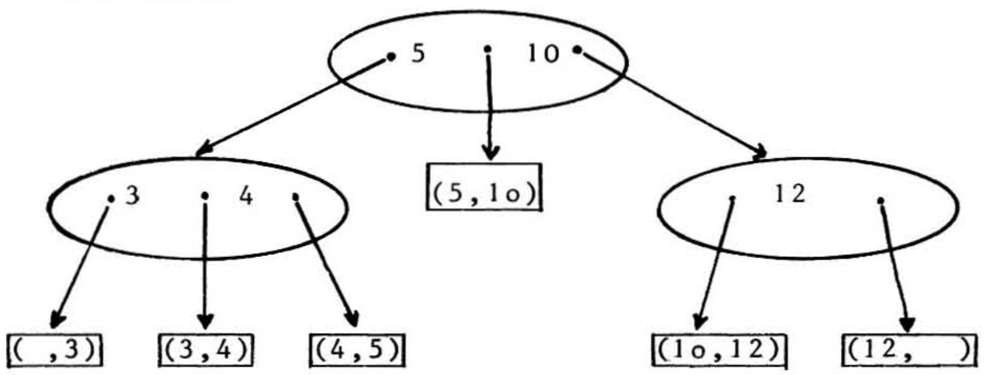
A preliminary version of this paper was presented at the 5th International Colloquium on Automata, Languages and Programming, Udine, Italy, July 17-21, 1978

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. Furthermore, an algorithm is described which constructs a nearly optimal code in linear time.

I. 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 towards the following problem. Consider the following ternary search tree. It has 3 internal nodes



and 6 leaves. The internal nodes contain the keys {3,4,5,1o,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 K1,K2 be the keys in node v;
           if X < K1 then SEARCH (X, left son of v)
           if X = K1 then exit (found);
           if K2 does not exist
           then SEARCH (X, right son of v)
           else begin if X < K2 then SEARCH (X, middle son of v);
           if X = K2 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 follow the pointer to the second subtree and it is cheaper to locate K_1 than to locate K_2.

We will also assume that the probability of access is given for each key and each interval between keys. More precisely, suppose we have n keys $B_1,\dots,B_n$ out of an ordered universe with $B_1 < B_2 < \dots < B_n$. Then β_i denotes the probability of accessing B_i, $1 \leq i \leq n$, and α_j denotes the probability of accessing elements X with $B_j < X < B_{j+1}$, $0 \leq j \leq n$. α_o 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 will always write the distribution of access probabilities as $\alpha_o,\beta_1,\alpha_1,\dots,\beta_n,\alpha_n$.

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

In our example, t = 2. The code word corresponding to 4, denoted W_2 is $a_o\ a_3$. The code word corresponding to (1o, 12), denoted V_4 is $a_4\ a_o$.

In general, a search tree is a prefix code

$C = \{V_o, W_1, V_1, \ldots, W_n, V_n\}$ with

$$V_j \in \Sigma^* \qquad\qquad W_i \in \Sigma^*\Sigma_{end}$$

where $\Sigma = \{a_o, a_2, a_4, \ldots, a_{2t}\}$ and $\Sigma_{end} = \{a_1, a_3, \ldots, a_{2t-1}\}$, $0 \leq j \leq n$, $1 \leq i \leq n$. 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}).

<u>Remark:</u> In the binary case, t = 1, letters a_o, 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 save synchronizing purposes. (cf. the example of an alphabetic Morse code at the end of section III).

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.

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

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

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

Remark: In the binary equal cost case ($t = 1$, $c_o = c_1 = c_2 = 1$) this definition coincides with the definitions of weighted path length used in the literature [e.g. Bayer, Itai, Knuth, Mehlhorn].

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 non-alphabetic 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 $a_o,\ldots,a_s$ and their costs $c_o,\ldots,c_s$ 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 non-alphabetic case and $\alpha_o,\beta_1,\ldots,\beta_n,\alpha_n$ in the alphabetic case. This should help the reader keeping things apart.

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

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

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

$$(2)\quad \mathrm{Cost}\,(C_{opt}) \le H/d + (\Sigma\alpha_j)[1/d + \max_{k \text{ even}} c_k] + (\Sigma\beta_i)\,[\max_{k \text{ odd}} c_k]$$

Note that lower and upper bound differ essentially by ln $\mathrm{Cost}(C_{opt})$. Inequality (1) is proved in Corollary 3. Theorem 2 gives a better bound than Corollary 3 but the bound is harder to state. Inequality (2) is proved in Theorem 4 by explicit construction of a code C satisfying (2). Moreover, this code can be constructed in linear time $O(t\cdot n)$ (Theorem 5).

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 the introduction by briefly reviewing some results.

Case 1: Equal letter costs; i.e. $c_i = 1$ for all i, $0 \leq i \leq s$.

In the nonalphabetic case an algorithm for the construction of an optimal code dates back to Huffmann; it can be implemented to run in time $O(n \log n)$ [van Leeuwen]. The noiseless coding theorem [Shannon] gives bounds for the cost of the optimal code, namely

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

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

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

$$H(\alpha_o,\beta_1,\ldots,\beta_n,\alpha_n) \leq \mathrm{Cost}(C_{opt}) + (\log e) - 1 + \log \mathrm{Cost}\ (C_{opt})$$

$$\mathrm{Cost}\ (C_{opt}) \leq H(\alpha_o,\beta_1,\ldots,\beta_n,\alpha_n) + 1 + \Sigma\alpha_j$$

Various approximation algorithms exist which construct codes in linear time in the binary case. The cost of these codes lie within the above bounds [Bayer, Mehlhorn, Fredman].

Case 2: Equal Probabilities

i.e. $p_i = 1/n$ for $1 \leq i \leq n$. The problem was solved by Perl , Garey and Even. 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 for the cost of an optimal code do exist.

Case 3: Unequal Probabilities, Unequal Letter Costs

This case was treated by Karp. 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 $\leq$ m) is NP-complete. A-priori bounds were proved by Krause, Csiszar and Cot.

The alphabetic case was treated by Itai. 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.

II. The Lower Bound

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

II. 1 The non-alphabetic case

II. 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. (Fig. 1).

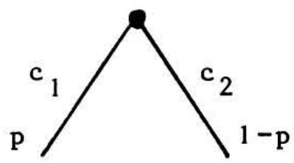


Figure 1

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

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

The following fact shows that G(p) is maximal for

$$p = 2^{-cc_1}, \; 1-p = 2^{-cc_2} \text{ where c is chosen such that}$$

$$2^{-cc_1} + 2^{-cc_2} = 1. \text{ So } G(p) \leq c \text{ for all p}$$

and

$$G(2^{-cc_1}) = c.$$

<u>Fact</u> (cf. e.g. Ash)

Let x_i, $y_i \geq 0$ for $1 \leq i \leq n$, $\Sigma x_i = 1 = \Sigma y$. Then

$$- \Sigma x_i \log x_i \leq - \Sigma x_i \log y_i.$$

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

II. 1.2 The Lower Bound in the Non-alphabetic 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 $\{a_o,\dots,a_s\}$. Let $c_i > 0$ be the cost of a_i, $0 \le i \le s$. Let c be such that

$$\sum_{i=o}^{s} 2^{-cc_i} = 1.$$

a) [Krause] $\quad \text{Cost}(C) \ge H(p_1,\dots,p_n)/c$

where $H(p_1,\dots,p_n) = -\Sigma p_i \log p_i$ is the entropy of the frequency distribution.

b) Let $h \in \mathbb{R}$, $h \ge o$ and

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

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

Remark: Inequality a) reads in its full form

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

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 than 2^{-h}.

Proof: a) Let $U_i = a_{i_1} a_{i_2} \dots a_{i_{\ell_i}}$. Define

$$q_i := \prod_{k=1}^{\ell_i} 2^{-cc_{i_k}}, \quad 1 \le i \le n.$$

$$Q := \sum_{i=1}^{n} q_i .$$

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

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

and hence by the fact above

$$\begin{aligned} H(p_1, \ldots, p_n) &= - \Sigma p_i \log p_i \\ &\leq - \Sigma p_i \log (q_i/Q) \\ &= c \text{ Cost } (C) + \log Q \\ &\leq c \cdot \text{Cost } (C) \end{aligned}$$

b) Let $h \geq 0$ and

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

Then

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

□

II.2 The alphabetic case

Every alphabetic code $C = \{V_o, W_1, \ldots, W_n, V_n\}$ is a non-alphabetic code and hence Theorem 1 applies. It shows

$$\text{Cost}(C) \geq 1/c \cdot H(\alpha_o, \beta_1, \ldots, \beta_n, \alpha_n)$$

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

$$\text{Cost}(C) \geq 1/d \cdot [H(\alpha_o, \beta_1, \ldots, \beta_n, \alpha_n) - \frac{c}{u} \cdot \max_{i \text{ odd}} c_i \cdot \ln H(\alpha_o, \beta_1, \ldots, \beta_n, \alpha_n)]$$

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

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

The alphabetic case differs from the non-alphabetic 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 only use 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 $1 \leq x < \infty$... be arbitrary, let

$$\tilde{c}_i = c_i \qquad \text{for } i \text{ even}$$

$$\tilde{c}_i = x \cdot c_i \qquad \text{for } i \text{ odd}$$

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

<u>Remark:</u> 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 one and hence $c(1) = c$, for $x = \infty$ the cost of letters in Σ_{end} is infinite and hence $c(\infty) = d$.

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

<u>Lemma 1:</u> $\widetilde{Cost}(C) \leq Cost(C) + (x-1) \cdot B \cdot \max_{i \text{ odd}} c_i$ for every x, $1 \leq x \leq \infty$, $B = \sum_{i=1}^{n} \beta_i$.

<u>Proof:</u> For $W_i \in \Sigma^* \Sigma_{end}$ let

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

Then
$$\widetilde{\mathrm{Cost}}(W_i) = \widetilde{\mathrm{Cost}}(W_i') + \tilde{c}_{j_i}$$
$$= \mathrm{Cost}(W_i') + x \cdot c_{j_i}$$
$$= \mathrm{Cost}(W_i) + (x-1)c_{j_i}$$

Hence
$$\widetilde{\mathrm{Cost}}(C) = \Sigma\beta_i \, \widetilde{\mathrm{Cost}}(W_i) + \Sigma\alpha_j \, \widetilde{\mathrm{Cost}}(V_j)$$
$$\leq \mathrm{Cost}(C) + (x-1)\cdot B \max_{i \text{ odd}} c_i$$

□

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

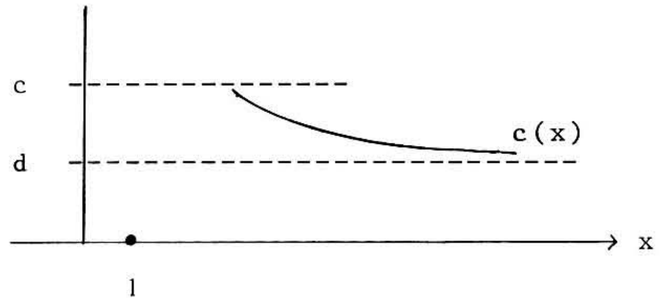
<u>Theorem 2:</u> Let $c(x)$ be such that $\sum_{k=0}^{2t} 2^{-c(x)\tilde{c}_k} = 1$

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

<u>Proof:</u> By Theorem 1,
$$\widetilde{\mathrm{Cost}}(C) \geq H(\alpha_o, \beta_1, \ldots, \beta_n, \alpha_n)/c(x)$$

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 $1 \leq x \leq \infty$. Write $c(x) = d + \delta(x)$.



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

$$H \leq c(x) \cdot \mathrm{Cost}(C) + (x-1) \cdot c(x) \cdot B \max_{i \text{ odd}} c_i$$

$$\leq d \cdot \mathrm{Cost}(C) + \delta(x) \cdot \mathrm{Cost}(C) + (x-1) \cdot c \cdot B \cdot \max_{i \text{ odd}} c_i$$

$$\leq d \cdot \mathrm{Cost}(C) + v \cdot e^{-u(x-1)} \cdot \mathrm{Cost}(C) + (x-1) \cdot c \cdot B \cdot \max_{i \text{ odd}} c_i$$

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

The right hand side is minimal (differential calculus) for $(x-1) = (\ln[u \cdot v \ \mathrm{Cost}(C)/c \cdot B \cdot \max_{i \text{ odd}} c_i])/u$

Hence

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

Using finally $y \ln 1/y \leq 1/e$ for all $y>0$ (in particular $y = \frac{cB \max c_i}{u}$) we obtain

<u>Corollary 3:</u> Let C be an alphabetic code for distribution $\alpha_o, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n$ with respect to costs $c_o, c_1, \ldots, c_{2t}$.
Let c,d be such that:

$$\sum_{k=o}^{2t} 2^{-cc_k} = 1 \qquad\qquad \sum_{k=o}^{t} 2^{-dc_{2k}} = 1$$

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

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

<u>Proof:</u> By the preceeding argument. □

Corollary 3 shows that the lower bound for the alphabetic code is essentially the lower bound ($d \cdot \mathrm{Cost}(C)$) for the non-alphabetic code where only the letters of even index are used plus a small correction of order ($c \cdot B \cdot \max_{i \text{ odd}} c_i \ln \mathrm{Cost}(C)$) which reflects the restricted usage of the letters in Σ_{end}.
A special case of Theorem 2 and Corollary 3 was proved by Bayer. He considered the binary alphabetic case with equal letter costs, i.e. $t = 1$ and $c_o = c_1 = c_2 = 1$.

It remains to prove Lemma 2. We will only show the existence of constants u,v but not derive a bound for them. This is justified since we recommend to always use Theorem 2 and to compute the maximal value of the right hand side by numerical methods. Corollary 3 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

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

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 $f(z,\delta(x)) \le 1$ 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

$$\text{(I1)} \quad \sum_{k=o}^{t} 2^{-(d+\delta(x))c_{2k}} + \sum_{k=1}^{t} 2^{-(x+\delta(x))zc_{2k-1}} \le 1$$

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 (I1) only increases the left hand side. It therefore suffices to show

$$\text{(I2)} \quad 2^{-\delta(x)c_{min}} \cdot \sum_{k=o}^{t} 2^{-dc_{2k}} + t2^{-dzc_{min}} \le 1$$

for some constants u,v. Using $\sum_{k=o}^{t} 2^{-dc_{2k}} = 1$ the left hand side of I2 is of the form

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

with $b_1 = 2^{c_{min}} > 1$, $b_2 = t\,2^{-dc_{min}} > 0$, $b_3 = (dc_{min} \ln 2)/u > 0$ 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

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

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

III. 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 [Gilbert and Moore, Mehlhorn].

The code is constructed top-down by repeated splitting of the ordered set $\{\alpha_o, \beta_1, \alpha_1, \ldots, \alpha_{n-1}, \beta_n, \alpha_n\}$ of probabilities. In each step we try to split the set as described in II.1.1 . Let d be such that

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

and let $s_{-1} = -\infty$, $s_{n+1} = \infty$

$$s_o = \alpha_o/2$$

$$s_i = \alpha_o + \beta_1 + \ldots + \beta_i + \alpha_i/2 \qquad \text{for } 1 \le i \le n.$$

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

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

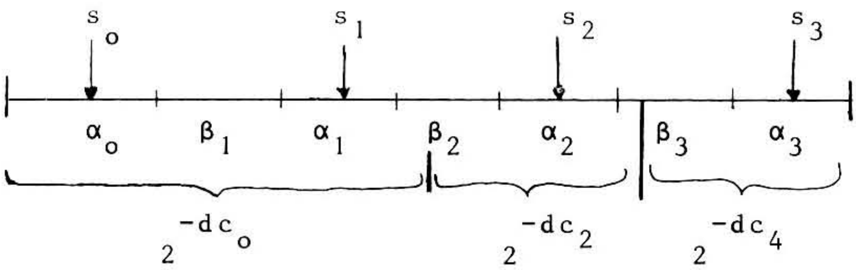


Fig. 2

From Fig. 2, it looks reasonable to assign letter a_o to $\alpha_o, \beta_1, \alpha_1$, 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 $W_2 = a_1$, $V_2 = a_2$, $W_3 = a_3$, $V_3 = a_4$ and let V_o, W_1, V_1 start with a_o. Next we have to work on the subproblem $\{\alpha_o, \beta_1, \alpha_1\}$. We split the interval $[0, 2^{-dc_o}]$ in the same way and obtain Fig. 3

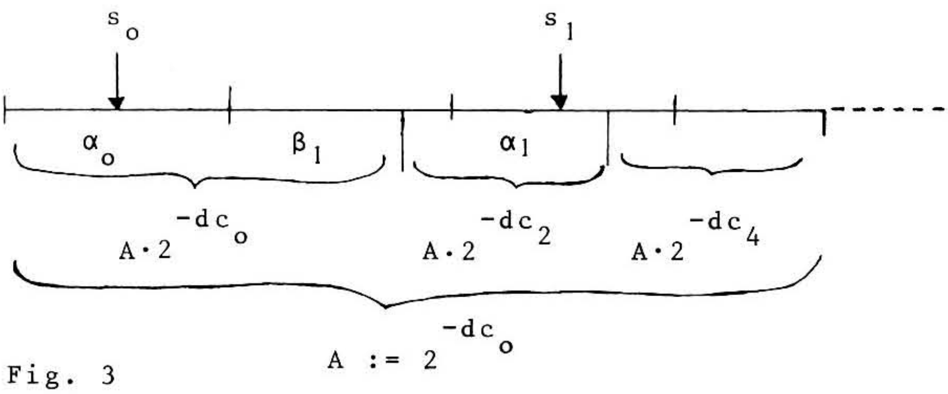


Fig. 3

This suggests to use letter a_o (a_1, a_2) as the second letter of the code words assigned to α_o (β_1, α_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

ℓ, r we work on the subproblem $\alpha_\ell, \beta_{\ell+1}, \ldots, \beta_r, \alpha_r$; $\ell \le r$

(1) L, R $L, R \in \mathbb{R}$, $L \le s_\ell \le s_r \le R$

U $U \in \Sigma^* = \{a_o, a_2, \ldots, a_{2t}\}^*$. U is a common prefix of code words V_ℓ, $W_{\ell+1}$, $V_{\ell+1}, \ldots, W_r$, V_r and

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

Initially $\ell = 0$, $r = n$, $L = 0$, $R = 1$ and $U = \varepsilon$ 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.

<u>Case 1:</u> $\ell = r$: Then we define $V_r = U$ and return

<u>Case 2:</u> $\ell < r$. We split the interval (L,R) in the ratio $2^{-dc_o} : 2^{-dc_1} : \ldots : 2^{-dc_{2t}}$. The i-th subinterval, $0 \le i \le t$, has boundaries $L_i = L + (R-L) \cdot \sum_{k=o}^{i-1} 2^{-dc_{2k}}$ 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 i-th interval.

If $h \le j$, i.e. some s_k's actually lie in the i-th subinterval, then we call procedure CODE recursively with parameters

$\ell = h$, $r = j$, $L = L_i$, $R = R_i$, $U = Ua_{2i}$

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 $L_o \le s_o \le \ldots \le s_4 < L_1 < L_2 < s_5 \le \ldots \le s_7 \le L_3 < s_8 \le R_3$. Then the recursive calls are CODE$(0,4,L_o,L_1,Ua_o)$, 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 by Fig. 4.

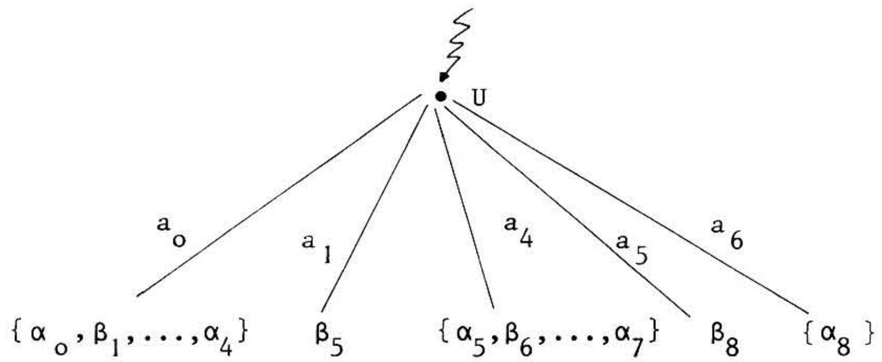


Fig. 4.

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 $a_{k_i} \in \Sigma_{end}$. The word W_i was constructed by the procedure CODE with actual parameters ℓ, r, L, R, U where $\ell < i \leq r$. Hence

$$\beta_i \leq \alpha_\ell/2 + \beta_{\ell+1} + \alpha_{\ell+1} + \ldots + \beta_q + \alpha_r/2$$

since β_i appears in that sum

$$= s_r - s_\ell$$

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

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

$$\mathrm{Cost}(W_i) \leq \mathrm{Cost}(U) + \max_{K\ \mathrm{odd}} c_K$$

$$\leq \frac{1}{d}\ [-\log\ \beta_i] + \max_{K\ \mathrm{odd}} c_K$$

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 $\ell < r$, $\ell \leq j \leq r$ and $V_j = Ua_{k_j}$ for some $a_{k_j} \in \Sigma$. Hence

$$\alpha_j/2 \leq \alpha_\ell/2 + \beta_{\ell+1} + \alpha_{\ell+1} + \ldots + \beta_r + \alpha_r/2$$

$$= s_r - s_\ell \leq R - L = 2^{-d\ \mathrm{Cost}(U)}$$

by the same reasoning as above. Hence

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

We summarize

<u>Theorem 4:</u> Let $(\alpha_o, \beta_1, \ldots, \beta_n, \alpha_n)$ be a probability distribution, $\beta_i \ge 0$, $\alpha_j \ge 0$, $\Sigma\beta_i + \Sigma\alpha_j = 1$.
Let $a_o, a_1, \ldots, a_{2t}$ be $(2t+1)$ symbols with costs $c_o, c_1, \ldots, c_{2t} \in \mathbb{R}_+$.

Then procedure CODE constructs an <u>alphabetic</u> code with

a) $\mathrm{Cost}\ (W_i) \le [-\log\ \beta_i]/d + \max_{k\ \text{odd}} c_k$

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

c) $\mathrm{Cost}\ (C) \le H(\alpha_o, \beta_1, \alpha_1, \ldots, \beta_n, \alpha_n)/d\ +$

$$(\Sigma\alpha_j)\ [1/d + \max_{k\ \text{even}} c_k]\ +$$

$$(\Sigma\beta_i)[\max_{k\ \text{odd}} c_k]$$

<u>Proof</u> 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. □

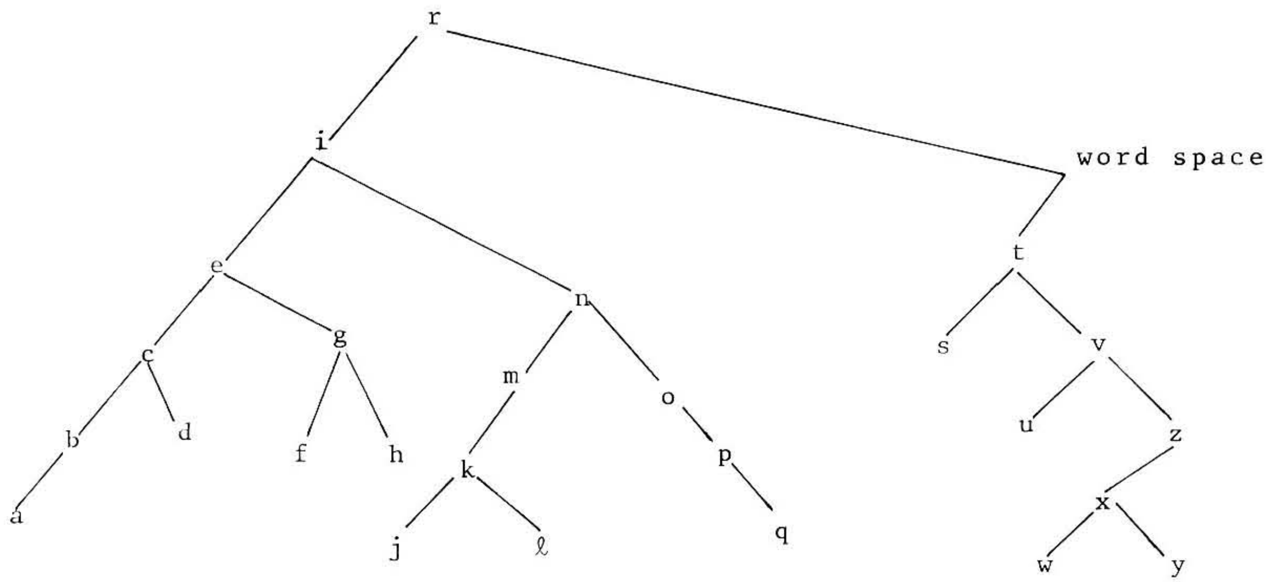
<u>Example:</u> An ordered Morse code. The Morse code is over a three letter alphabet: dot (cost 1), dash (cost 2) and letter space (cost 1). We assume the ordering dot < letter space < dash i.e. $\Sigma = \{\text{dot, dash}\}$ and $\Sigma_{end} = \{\text{letter space}\}$.

Then $c_o = 1$, $c_1 = 1$, $c_2 = 2$, $2^{-d} = 0.618$ 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,..., β_{27} = probability of word space. We refer the reader to [Bauer, Goos] for the exact values of $\beta_1, \beta_2, \ldots, \beta_{27}$. All α_j's are zero.
Then $H(\alpha_o, \beta_1, \ldots, \beta_{27}, \alpha_{27}) = 4.1$. The lower bound of Theorem 2 is

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

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 4 is 5.85 . The code actually constructed is



, i.e. r is encoded by letter space, i is encoded by dot letter space, n by dot dash letter space. The cost of this code is 4.3o25. In comparison, the cost of the morse code is 4.o55. The morse code is non-alphabetic.

IV 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

$$z_i = \sum_{k=0}^{i} 2^{-dc_{2k}} \quad \text{for } 0 \leq i \leq t.$$

Procedure CODE has the following global structure.

```
procedure CODE(ℓ,r,L,R,U);
begin
if ℓ= r
then V_ℓ ← U
else begin
(*)         for all i, 0 ≤ i ≤ 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} ≤ L_i < s_h and s_j ≤ R_j < s_{j+1};
                   if h ≤ j then CODE(h,j,L_i,R_i,Ua_{2i});
                   if j+1 ≤ r then W_{j+1} ← 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, $\ell \leq i \leq r$, lie in the same subinterval. Note that problem c) does not affect the analysis given in section III, however it will affect running time.

Consider problem b) first. We describe a solution for the 0-th subinterval. By definition $L_o = L$ and hence $s_{\ell-1} \leq L_o \leq s_\ell$ by assumption. Hence we only have to find j such that $s_j \leq R_o < s_{j+1}$ We find j by exponential + binary search [Fredman] . We first compare R_o with

$$s_{\ell+1}, \; s_{\ell+2}, \; s_{\ell+4}, \; s_{\ell+8} \qquad \text{until}$$

$$s_{\ell+2^k} > R_o \quad \text{or} \quad \ell + 2^k > r$$

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

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

Since $j - \ell \geq 2^{k-1}$ where k is determined as above it follows that j can be determined in time $\leq a(1 + \log(n_o+1))$ where a is a suitable constant.

Next we address problem a). Let n_i, $0 \leq i \leq t$, be the number of s_i's which lie in the i-th interval. The obvious way to proceed is to determine $n_o, n_1, n_2, \ldots, n_t$ 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_o and n_t in parallel, stop when anyone of them is found. Say n_o was determined first. Forget everything about n_t. Now determine n_1 and n_t in parallel ... .

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

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

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 0-th subinterval (if $j \geq 1$) or the rightmost probability α_r to the t-th subinterval (if $j < t$). More precisely, suppose $j \geq 1$. Then we set $V_\ell \leftarrow Ua_o$, $W_{\ell+1} \leftarrow Ua_1$ and call CODE recursively with parameters $\ell+1$, r, L_j, R_j, Ua_{2j}. Note that the analysis of section III 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 $(\alpha_o,\beta_1,\dots,\beta_n,\alpha_n)$ and code alphabet $a_o,a_1,\dots,a_{2t-1}, a_{2t}$ with costs $c_o,c_1,\dots,c_{2t}$. Note that n+1 is equal to the number of α_j's.

Then

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

for some constant a.

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

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

Since n_i is the number of s_j's which fall in the i-th subinterval we have $n+1 = n_o + n_1 + \dots + n_t$. Also $0 \leq n_i$ and $n_i \leq n$ 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)

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

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

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

Here a is some constant; w.l.o.g. we can use the same a in all equations.

<u>Theorem 5:</u> $T(n,t) = O((t+1)\cdot n)$

<u>Proof:</u>

We show by induction on n

(*) $\qquad T(n,t) \le d(t+1)\cdot n - e(t+1)\cdot\log(n+1)$

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

Induction base: $n = 0$, $n = 1$ or $n = n_o+\ldots+n_t$,
$0 \leq n_i < n$, $\max n_i \leq 4$ and $T(n,t) = T_2(n,t)$. Then

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

and

$$T(n,t) \leq a(t+1)\cdot(\text{number of } n_i\text{'s} \neq 0) + a(t+1)(1+\log 5)$$
$$\leq a(t+1)\cdot n + a(t+1) \log 10$$

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

Induction step: Let $n = n_o+\ldots+n_t$, $0 \leq n_i < n$, $\max n_i > 4$ and $T(n,t) = T_1(n,t)$. Then by induction hypothesis

$$T(n,t) \leq \sum_{i=0}^{t} [d(t+1)n_i - e(t+1) \log(n_i+1) + a(1+\log(n_i+1)] - \max_{0 \leq i \leq t} a(1+\log(n_i+1))$$

We may assume w.l.o.g. that $n_o = \max n_i$.

Then

$$T(n,t) \leq d(t+1)\cdot n - e(t+1) \log(n+1)$$
$$+ e(t+1) \log(n+1) + \sum_{i=1}^{t} a(1+\log(n_i+1)) - \sum_{i=0}^{t} e(t+1)\log(n_i+1)$$

It suffices to show

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

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

$$e(t+1)\log(n+1)+at \leq e(t+1)\log(n_o+1)+(e(t+1)-a)\log(n-n_o+1)$$

The derivative of the right hand side with respect to n_o is

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

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

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

or

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

since $n > n_o \geq 5$ one only has to choose e such that

$$\log 7/6 \leq (e-a)/e$$

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

$$e(t+1) \log \frac{n+1}{n_o+1} + at \leq (e(t+1)-a) \log(\frac{t}{t+1} n+1)$$

Since $t \geq 1$, $(n+1)/(n_o+1) \leq t+1$ and $tn/(t+1)+1 \geq 5t+1 = 5(t+1)-4$ it suffices to show

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

or

$$a(t+\log(5(t+1)-4)) \leq e(t+1)\cdot\log \frac{5(t+1)-4}{t+1}$$

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

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

for $t \geq 1$.

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

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

or

$$e(t+1)\log\frac{n+1}{n-4} + a \log(n-4) \leq e(t+1)\log 6 - at$$

Since $5 = n_o < n < 5(t+1)$ it suffices to show

$$e(t+1) \log 7/2 + a \log 5t \leq e(t+1) \log 6 - at$$

or

$$a(t+\log 5t) \leq e(t+1) \log 12/7$$

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

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

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

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

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

Theorem 5 shows that a prefix code satisfying the inequality of Theorem 4 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 [Altenkamp, Mehlhorn].

Variant A:

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

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

Variant B:

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

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

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

B i b l i o g r a p h y

ALTENKAMP & MEHLHORN: Codes: Unequal Probabilities, Unequal Letter Costs, Techn. Bericht, Fachbereich 10, Universität des Saarlandes, 6600 Saarbrücken, A 77/13

ASH: Information Theory, Interscience Publishers, N.Y., 1965

BAUER & GOOS: Informatik, Heidelberger Taschenbücher, Springer-Verlag, 1971

BAYER: Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees, to appear in Acta Informatica

CSISZAR: Simple proofs of some theorems on noiseless channels, Inf. and Control 14, pp. 285-298, 1969

COT: Characterization and Design of Optimal Prefix Codes, Ph.D. Thesis, Stanford University, June 1977

FREDMAN: Two Applications of a Probabilistic Search Technique, ACM, Conf. on Theory of Computing, 1975

GILBERT & MOORE: Variable Length Encodings, Bell System Technical Journal, 38 (1959), 933-968

HU & TUCKER: Optimal Search Trees and Variable Length Alphabetic Codes, SIAM J. Appl. Math. 21, 1971, 514-532

HUFFMANN: A Method for the Construction of Minimum-Redundancy Codes, Proc. IRE 4o, 1o98-11o1, 1952

ITAI: Optimal Alphabetic Trees, SIAM J. Computing, (5), 1976, 9-18

KARP: Minimum Redundancy Coding for the Discrete Noiseless Channel

KNUTH: Optimum Binary Search Trees, Acta Informatica, 1, 1971, pp. 14-25

KRAUSE: Channels which transmit letters of unequal duration, Inf. and Control 5, pp. 13-24, 1962

MEHLHORN: Effiziente Algorithmen, Teubner Studienbücher Informatik, Stuttgart 1977.

MEHLHORN: Best Possible Bounds on the Weighted Path Length of Optimum Binary Search Trees, SIAM J. of Comp., Vol. 6, No 2, 1977, 235-239

PERL, GAREY & EVEN: Efficient Generation of Optimal Prefix Code : Equiprobable Words Using Unequal Cost Letters, JACM, Vol. 22, No 2, 2o2-214, April 1975

VAN LEEUWEN: On the construction of Huffmann trees, 3rd ICALP (1976), 382-41o, Ed. S. Michaelson and R. Milner, Edinburgh University Press