An efficient algorithm for constructing nearly optimal prefix codes

by

Kurt Mehlhorn

Fachbereich 10 -
Angewandte Mathematik
und Informatik
Universität des Saarlandes
6600 Saarbrücken
West Germany

September 1978

A 78/13

Abstract: A new algorithm for constructing nearly optimal prefix codes in the case of unequal letter costs and unequal probabilities is presented. A bound on the maximal deviation from the optimum is derived and numerical examples are given. The algorithm has running time $O(t \cdot n)$ where t is the number of letters and n is the number of probabilities.

I. Introduction

We study the construction of prefix codes. Given is a set $p_1, p_2, \ldots, p_n$ of probabilities, $p_i > 0$ and $\sum_{i=1}^{n} p_i = 1$,

and a set $a_1, \ldots, a_t$ of letters; letter a_i has cost $c_i \in \mathbb{R}$, $c_i > 0$. A prefix code T over the alphabet $\Sigma = \{a_1, a_2, \ldots, a_t\}$ is a set $U_1, \ldots, U_n$ of words in Σ^* such that no U_i is a prefix of any U_j for $i \neq j$. Let

$$U_i = a_{j_1} a_{j_2} \ldots a_{j_{\ell_i}}$$

be the i-th code word. Its cost $C(U_i)$ is defined as the sum of the letter costs, i.e.

$$C(U_i) = c_{j_1} + c_{j_2} + \ldots + c_{j_{\ell_i}} \quad .$$

Finally, the average cost of code T is defined as

$$C(T) = \sum_{i=1}^{n} p_i \; C(U_i).$$

At present, there is no efficient algorithm for constructing an optimal (= minimum average cost) code given $p_1, \ldots, p_n$ and $c_1, \ldots, c_t$. Karp formulated the problem as an integer programming problem and hence his algorithm may have exponential time complexity. Various approximation algorithms are described in the literature [Krause [1], Csiszar [2], Altenkamp and Mehlhorn [3] , Cot [4]. They construct codes T such that

$$H(p_1, \ldots, p_n) \leq c \cdot C_{opt} \leq c \cdot C(T) \leq H(p_1, \ldots, p_n) + $$
$$+ f(c_1, c_2, \ldots, c_t) + \gamma$$

where $H(p_1, \ldots, p_n) = -\Sigma p_i \log p_i$ is the entropy of the probability distribution, c is defined such that $\sum_{i=1}^{t} 2^{-cc_i} = 1$ (root of characteristic equation of letter costs), C_{opt} is the cost of an optimal code, $f(c_1,...,c_t)$ is some function of the letter costs and γ is a small constant. In most cases (Krause [1], Csiszar [2], Altenkamp and Mehlhorn [3]) $f(c_1,...,c_t) = \max\{c_i \mid 1 \le i \le t\}$ while for Cot [4] $f(c_1,...,c_t)$ is a more complex function.

Here we describe another approximation algorithm and prove a similar bound for the cost of the code constructed by it (section II). In section III we indicate that our algorithm has linear running time $O(t.n)$ and report some experimental results. They suggest that the new algorithm constructs better codes than the previous algorithms.

II. The Algorithm and its Analysis

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)

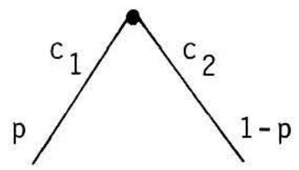

Figure 1 : Splitting a set into two parts.

The local information gain per unit cost is then

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

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

$$G(p) = \frac{-p \log p - (1-p) \log (1-p)}{(-p\cdot\log 2^{-cc_1} - (1-p) \log 2^{-cc_2})\cdot\frac{1}{c}}$$

for all $c \neq 0$

It is easy (elementary calculus) to see 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 $G(p) \leq c$ for all p and $G(2^{-cc_1}) = c$.

The argument above suggests the following approximation algorithm: try to split the given set of probabilities into two parts of probability p and 1-p respectively so as to make $p-2^{-cc1}$ as small as possible. Such a split maximizes the local information gain per unit cost and should (hopefully) produce a good prefix code. For the sake of efficiency our algorithm only considers splits of the form $\{p_1,\ldots,p_i\}$, $\{p_{i+1},\ldots,p_n\}$.

Next we illustrate the approach by an example. Given are probabilities $(p_1,p_2,\ldots,p_6) = (.3,.1,.05,.25,.2,.1)$ and the code alphabet a_1,a_2 with costs $(c_1,c_2) = (1,2)$. We choose c such that $2^{-cc_1} + 2^{-cc_2} = 1$. Then $2^{-cc_1} = 0.618$.

We draw the probabilities $p_1,\ldots,p_6$ as a partition of the unit interval and split the unit interval into pieces of length 2^{-cc_1} and 2^{-cc_2} respectively. (Fig. 2)

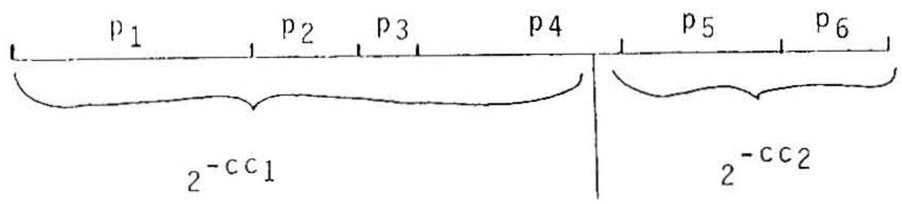

Fig. 2: Splitting the unit interval.

The split goes through the right half of p_4. So we assign letter a_1 to p_1,p_2,p_3 and p_4 and letter a_2 to p_5 and p_6 (Fig. 3).

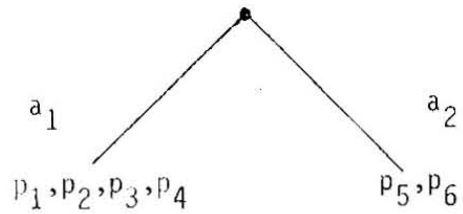

Fig. 3: The code tree after the first split.

Next we apply the same strategy to the set $p_1,\ldots,p_4$, i.e. we consider the interval p_1,p_2,p_3,p_4 and split it in the ratio 2^{-cc_1} to 2^{-cc_2} (Fig. 4).

Caution: At this point our approach differs from the one taken by Krause, Csizar and Altenkamp & Mehlhorn. After having split the unit interval into two parts in the first step, they split the interval of length 2^{-cc1} in the ratio 2^{-cc1} to 2^{-cc2} in the second step. Thus their approach can be viewed as a digital expansion process. We continue this remark after the precise definition of our new algorithm below.

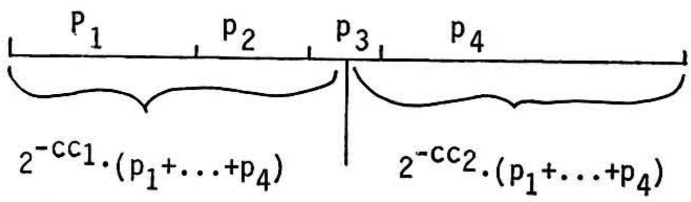

Fig. 4 : Splitting the interval $p_1,\ldots,p_4$

We proceed with our example. In Figure 4 the split goes through the right half of p_3. So we assign letter a_1 to p_1,p_2,p_3 and letter a_2 to p_4 (Fig. 5)

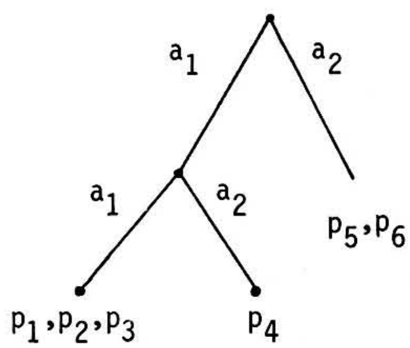

Fig. 5: The code tree after the second split

Proceeding in this fashion the following code will be constructed (Fig. 6).

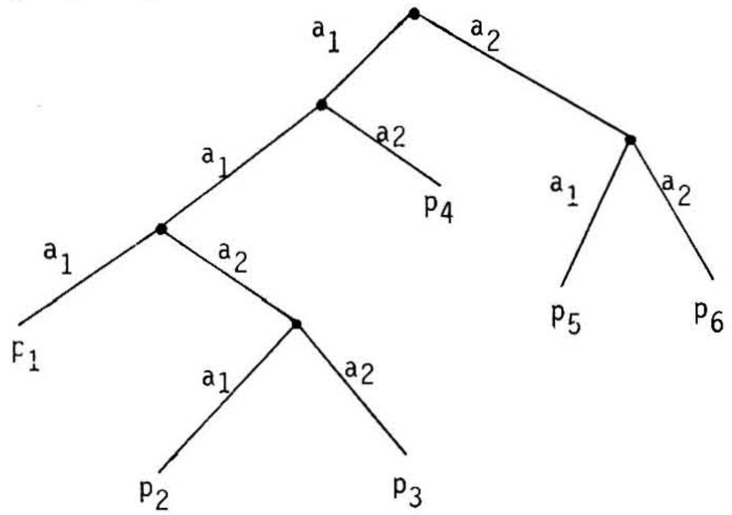

Fig. 6: The code constructed by the new algorithm described in this paper.

This code has cost

$$0.3 \cdot 3 + 0.1 \cdot 5 + 0.05 \cdot 6 + 0.25 \cdot 3$$
$$+ 0.2 \cdot 3 + 0.1 \cdot 4 \qquad = 3.45$$

So much for the intuitive description of the algorithm. For the precise definition by a pseudo-ALGOL program we need some notation

Let $c \in \mathbb{R}$ be such that $\sum_{j=1}^{t} 2^{-cc_j} = 1$. Then 2^{-c} is traditionally called the root of the characteristic equation of the letter costs.

Let $P_k = p_1+p_2+\ldots+p_k$, $0 \leq k \leq n$ and

$s_k = p_1+p_2+\ldots+p_{k-1} + p_k/2$, $1 \leq k \leq n$.

A call CODE $(1,n,\varepsilon)$ constructs a prefix code for the probability distribution $p_1,\ldots,p_n$. Here ε denotes the empty word over the alphabet $\{a_1,\ldots,a_t\}$.

procedure CODE(ℓ,r,U);

comment: ℓ and r are integers, $1 \le \ell \le r \le n$, and U is a word over $\{a_1,\ldots,a_t\}$. We will construct code words for $p_\ell, p_{\ell+1},\ldots,p_r$. The word U is a common prefix of code words U_ℓ, $U_{\ell+1},\ldots,U_r$.

begin

if $\ell = r$

then we take U as the code word U_ℓ

else begin $L \leftarrow P_{\ell-1}$; $R \leftarrow P_r$;

for m, $1 \le m \le t$ do

begin $L_m \leftarrow L + (R-L)\cdot \sum_{j=1}^{m-1} 2^{-cc_j}$;

$R_m \leftarrow L_m + (R-L)\cdot 2^{-cc_m}$;

$I_m \leftarrow \{i;\ L_m \le s_i < R_m\}$

end;

Comment: I_m, $1 \le m \le t$, is a (not necessarily non-trivial) partition of the set $\{\ell,\ldots,r\}$. Since we certainly do not want to assign the same letter to all probabilities, $p_\ell,\ldots,p_r$, we need to make sure that the partition is non-trivial. The easiest way to ensure non-trivialty is to force the use of letters a_1 and a_t, i.e. to make I_1 and I_t non-empty;

if $I_1 = \emptyset$

then begin let m be minimal with $I_m \neq \emptyset$;

$I_1 \leftarrow \{\ell\}$; $I_m \leftarrow I_m - \{\ell\}$;

end;

if $I_t = \emptyset$

then begin let m be maximal with $I_m \neq \emptyset$;

$I_t \leftarrow \{r\}$; $I_m \leftarrow I_m - \{r\}$;

end;

<u>comment</u> whenever we refer to partition I_m, $1 \leq m \leq t$, outside the definition of CODE, then the partition is meant, as it exists at this point of the program;

<u>for</u> m, $1 \leq m \leq t$ <u>do</u>

<u>if</u> $I_m \neq \emptyset$ <u>then</u> CODE (min I_m, max I_m, Ua_m)

<u>end</u>

<u>end</u>.

<u>Remark:</u> Procedure CODE is a generalization of Shannon's binary splitting algorithm [5] for constructing nearly optimal codes over a binary alphabet. It has been generalized in a different direction in the past by Krause, Csizar, Altenkamp & Mehlhorn, who view the binary splitting algorithm as a fractional expansion process

Consider the binary fraction $.x_1 x_2 \ldots x_m$ with $x_i \in \{0,1\}$. We can define the real number represented by that binary fraction recursively as

$\mathrm{Num}(x_m)$ = <u>if</u> $x_m = 0$ <u>then</u> 0 <u>else</u> 1/2

$\mathrm{Num}(x_i x_{i+1} \ldots x_m)$ =

<u>if</u> $x_i = 0$ <u>then</u> $0 + 1/2\ \mathrm{Num}(x_{i+1} \ldots x_m)$

<u>else</u> $1/2 + 1/2\ \mathrm{Num}(x_{i+2} \ldots x_m)$

So, binary fraction expansion corresponds to repeated splitting of the interval in the relation 1/2 : 1/2. Suppose now that we split instead in the relation 2^{-cc1} : $(1-2^{-cc1})$. Then we should define Num as follows.

$\mathrm{Strangenum}(x_m)$ = <u>if</u> $x_m = 0$ <u>then</u> 0 <u>else</u> 2^{-cc_1}

$\mathrm{Strangenum}(x_i x_{i+1} \ldots x_m)$ =

<u>if</u> $x_i = 0$ <u>then</u> $0 + 2^{-cc_1}\ \mathrm{Strangenum}(x_{i+1} \ldots x_m)$

<u>else</u> $2^{-cc_1} + (1-2^{-cc_1}) \cdot \mathrm{Strangenum}$
$(x_{i+1} \ldots x_m)$

We are now ready to take up the remark (labelled caution) and to outline the fractional expansion approach of our example. Consider the fractional expansions of reals $s_1, s_2, \ldots, s_6$ in our "strange number system". The first digit is 0 for s_1, s_2, s_3, s_4 and 1 for s_5 and s_6. Figure 7 in addition shows the second digits in the expansion of s_1, s_2, s_3, s_4.

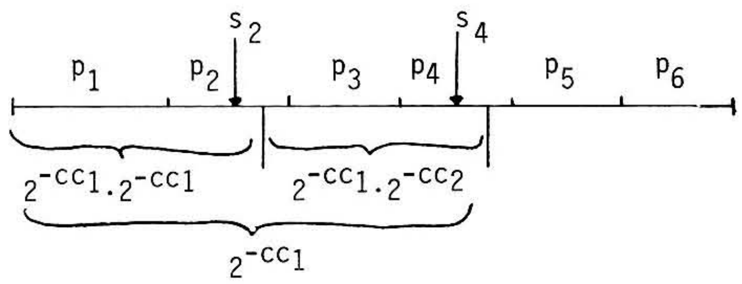

Figure 7 : The first two steps of the fractional expansion method

Note that 0 is the second digit in the expansions of s_1 and s_2 and 1 is the second digit in the expansions of s_3 and s_4. Proceeding in this fashion until a prefix code is obtained we will construct the code shown in Figure 8 of cost 3.75

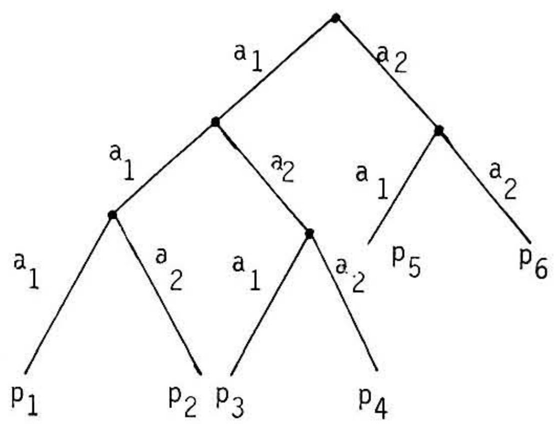

Figure 8: The code constructed by the fractional expansion method.

So much for the fractional expansion approach. The approach taken in this paper follows Shannon's ideas more closely. After having split the original set of probabilities into sets $\{p_1,p_2,p_3,p_4\}$ and $\{p_5,p_6\}$ in Fig. 1 we treat each subproblem in the same way as the original problem. This approach was studied before by Bayer [6] in the binary equal letter cost case, $t = 2$, $c_1 = c_2 = 1$. It generally yields much better codes (cf. the experimental results at the end of the paper).

In the remainder of this section we will prove the following theorem.

Theorem: Given probabilities $p_1,\dots p_n$ and letters $a_1,\dots,a_t$ of cost $c_1,\dots,c_t$ and a real c such that $\sum_{m=1}^{t} 2^{-cc_m} = 1$

procedure CODE constructs a code tree T of average cost C(T) with

$$c \cdot C(T) \le H(p_1,\dots,p_n) + 1 - p_1 - p_n + cc_{max}$$

where $c_{max} = \max\{c_m;\ 1 \le m \le t\}$.

Proof: The proof proceeds in two steps. We first derive a managable expression for the difference $c \cdot C(T) - H(p_1,\dots,p_n)$ and then derive a bound on that difference.

Procedure CODE constructs a code tree T for probabilities $p_1,\dots,p_n$. Let v be any node of the complete infinite tree over letters $a_1,\dots,a_t$ and let U be the word corresponding to node v, i.e. U is spelled along the path from the root to node v. Define

$$w(v) := \Sigma\{p_i;\ U \text{ is a prefix of code word } U_i \text{ for } p_m\}$$

and

$$w_m(v) := w(v_m)$$

where v_m corresponds to Ua_m. Then

$$w(v) = w_1(v) + w_2(v) + \ldots + w_t(v)$$

If v is an element of code tree T then let ℓ and r be the other two parameters in the call CODE(ℓ,r,U). Apparently,

$$w(v) = p_\ell + p_{\ell+1} + \ldots + p_r.$$

Let N_T be the set of interior nodes of code tree T.

Lemma 1:

1) The cost C(T) of code tree T is equal to

$$C(T) = \sum_{v \in N_T} \sum_{j=1}^{t} c_j \cdot w_j(v)$$

2) The entropy $H(p_1,\ldots,p_n)$ is equal to

$$H(p_1,\ldots,p_n) = \sum_{v \in N_T} w(v) \cdot H\left(\frac{w_1(v)}{w(v)}, \ldots, \frac{w_t(v)}{w(v)}\right)$$

Proof: The proofs are simple inductions of the depth of tree T. Note that 2) is just repeated application of the grouping axiom and 1) is essentially reordering of summation. In

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

we sum over the leaves of the code tree. If for every interior node v and letter a_j we consider those code words U_i which go through v and use letter a_j in node v then we obtain the summation formula given in the lemma. □

Lemma 1 allows us to write

$$(1) \qquad c \cdot C(T) - H(p_1,\dots,p_n) =$$

$$= \sum_{v \in N_T} \left[\sum_{m=1}^{t} cc_m \cdot w_m(v) - w(v) \cdot H\left(\frac{w_1(v)}{w(v)},\dots,\frac{w_t(v)}{w(v)}\right) \right]$$

$$= \sum_{v \in N_T} w(v) \left[\sum_{m=1}^{t} \frac{w_m(v)}{w(v)} \left(\log 2^{cc_m} + \log \frac{w_m(v)}{w(v)}\right) \right]$$

We now arrived at our expression for $c \cdot C(T) - H(p_1,\dots,p_n)$. In order to derive an upper bound on that difference we will try to bound

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

Lemma 2 gives us the necessary information about $w_m(v)/w(v)$.

Lemma 2: Consider any call Code (ℓ,r,U), let node v correspond to word U and let $\ell < r$. Let sets $I_1,\dots,I_m$ be defined as in procedure CODE. Then for $1 \le m \le t$

a) if $I_m = \emptyset$ then $w_m(v) = 0$

b) if $I_m = \{e\}$ then $w_m(v) = p_e$

c) if $|I_m| \ge 2$ and $e = \min I_m$, $f = \max I_m$, then for $2 \le m < t$

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

$$\frac{w_1(v)}{w(v)} \le 2^{-cc_1} + \frac{p_f}{2w(v)} \le 2\cdot 2^{-cc_1}$$

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

Proof: a) and b) are obvious. Consider c) now. Suppose first that $2 \le m < t$. Figure 9 shows the meaning of e and f.

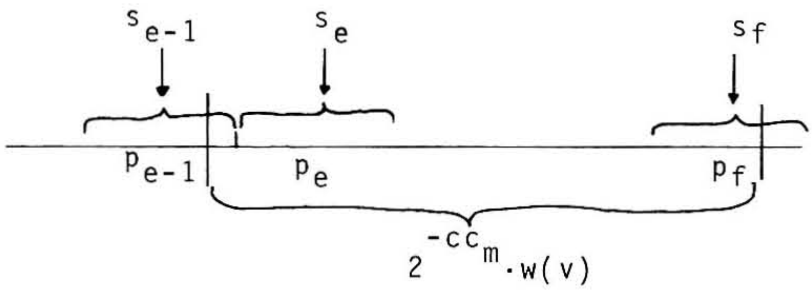

Figure 9 : A typical element of the partition.

Then $w_m(v) = p_e+p_{e+1} +\ldots+ p_{f-1}+p_f$ and

$p_e/2+p_{e+1} +\ldots+ p_{f-1}+p_f/2 \le 2^{-cc_m}\cdot w(v)$ by definition of $w(v)$,

$w_m(v)$ and I_m. Hence

$$w_m(v) - 2^{-cc_m}w(v) \le (p_e+p_f)/2 \le 2^{-cc_m}\cdot w(v)$$

If m = 1 then we even have

$$p_e+p_{e+1} +\ldots+ p_{f-1}+p_f/2 \le 2^{-cc_1}\cdot w(v)$$

and hence

$$w_1(v)-2^{-cc_1}\cdot w(v) \le p_f/2 \le 2^{-cc_1}\cdot w(v)$$

An analogous statement holds for m = t. □

We are now ready to derive an upper bound on $E(v,m)$ defined in equation (2) above.

Case a: $I_m = \emptyset$. Then $w_m(v) = 0$ and hence $E(v,m) = 0$.

Case b: $I_m = \{e\}$. Then $w_m(v) = p_e$ and $w_m(v)/w(v) \leq 1$. Hence

$$E(v,m) \leq \frac{w_m(v)}{w(v)} \cdot \log 2^{cc_m} = (cc_m \cdot p_e)/w(v)$$

Case c: $\{I_m\} \geq 2$. Let $e = \min I_m$, $f = \max I_m$.
Let $y := 2^{-cc_m}$ and $x := w_m(v)/w(v) - 2^{-cc_m}$.
Then $x \leq \frac{p_e+p_f}{2w(v)} \leq 2^{-cc_m}$ by Lemma 2. We may rewrite $E(v,m)$ as

$$\begin{aligned} E(v,m) &= (x+y)\,[\log 1/y + \log(x+y)] \\ &= (x+y) \log(1 + x/y) \end{aligned}$$

Lemma 3: Let $0 \leq x \leq y$ and $0 < y$. Then

$$(y+x) \cdot \log (1+x/y) \leq 2x$$

Proof: Consider

$$f(x) = 2x - (y+x) \log (1+x/y)$$

Then

$$\begin{aligned} f'(x) &= 2-\log(1+x/y) - \frac{(x+y)/y}{\ln 2\cdot(1+x/y)} \\ &= (2-1/\ln 2) - \log(1+x/y) \end{aligned}$$

Thus f' is monotonically decreasing and hence

$\min\{f(x);\ 0 \le x \le y\} = \min\{f(0),\ f(y)\} = 0$ □

From Lemma 3 we conclude

$$E(v,m) \le 2x = (p_e + p_f)/w(v)$$

for $m = 1$ we can even conclude $E(v,m) \le p_f/w(v)$ and for $m = t$ we conclude $E(v,m) \le p_e/w(v)$.

In either case we have now derived an upper bound on $E(v,m)$.

It remains to consider the problem how often a certain probability p_i can be used in the bounds of the different kind. First note that each probability is used exactly once in a bound corresponding to case b) of Lemma 3. Next suppose that p_i is used in a bound of kind c); say $i = \min I_m$. Then this will lead to a recursive call CODE(i,max I_m,). If $I_m = \{i\}$ then this is a terminal call of CODE and i will at most be used in a bound of kind b). If $|I_m| \ge 2$ then in the body of CODE (i,max I_m,) a partion of I_m will be defined. Call this partion J_k, $1 \le k \le t$. We will certainly have $i \in J_1$. Now note, that Lemma 2 states that for J_1 we don't have to use min J_1 in order to bound $E(v,m)$. Since i will always be in the first set of the partition for all further recursive calls of CODE, we conclude that i must only be used once in a bound of kind c).

In summary, we use each probability p_i at most once in a bound of kind b) and at most once in a bound of kind c). Furthermore the argument above shows that p_1 and p_n are never used in a bound of kind c).

We will now substitute the bounds on $E(v,m)$ into equation (1), our expression for the difference $c \cdot C(T) - H(p_1,\ldots,p_n)$.

The bounds of kind b) contribute at most

$c \cdot c_{max} \cdot \sum_{i=1}^{n} p_i = c \cdot c_{max}$ where $c_{max} = \max\{c_m;\ 1 \leq m \leq t\}$ and

the bounds of kind c) contribute at most $\sum_{i=2}^{n-1} p_i = 1 - p_1 - p_n$.

Hence

$$c \cdot C(T) - H(p_1,\ldots,p_n) \leq c \cdot c_{max} + 1 - p_1 - p_n$$

□

Note that among others, Krause has shown that $c \cdot C(T) \geq H(p_1, \ldots, p_n)$ for every prefix code T and hence procedure CODE constructs very good codes indeed.

III. Implementation and Experimental Data

Altenkamp and Mehlhorn describe an implementation of their algorithm which has running time $O(t \cdot n)$. The same methods can be used to implement procedure CODE such that its running time is $O(t \cdot n)$. We refer the reader to Altenkamp & Mehlhorn for details.
In Güttler et al.[7] the algorithms described in Altenkamp & Mehlhorn (which is very similar to the one described by Krause and Csiszar) and the algorithm described here were compared in the binary equal letter cost case, $t = 2$ $c_1 = c_2 = 1$. 200 examples were run; for each of them the optimal code was constructed. Fig. 10 shows the average and maximal values of $C_1/C_{opt} \cdot 100$ and $C_2/C_{opt} \cdot 100$ where C_{opt} is the cost of the optimal code, C_1 and C_2 are the costs of the code constructed by the algorithm described here and the algorithm described by Altenkamp and Mehlhorn respectively.

	C_1(procedure CODE above)	C_2(Altenkamp & Mehlhorn)
average value of $C/C_{opt} \cdot 100$	104.5	119.7
maximal value of $C/C_{opt} \cdot 100$	109.0	154.7

Fig. 10 : Experimental comparison of two algorithms

Cot describes yet another procedure for constructing nearly optimal prefix codes. He proves that the average cost C of his code satisfies

$$H(p_1,\dots,p_n)/c + \delta \le C \le H(p_1,\dots,p_n)/c + \delta + c_{min}$$

where

$$\delta = \sum_{i=2}^{t} c_i \log_{\lambda_t}(\lambda_i/\lambda_{i-1}) \text{ and } \sum_{j=1}^{i} 2^{-(\log \lambda_i)c_j} = 1, \text{ and}$$

$$c_{min} = \min\{c_i\}$$

for $1 \le i \le t$. He does not describe a detailled implementation of his algorithm nor does he estimate the running time of his algorithm. In our example $c_1 = c_2 = 1$, and hence $\lambda_1 = 1$, $\lambda_2 = 2$, $c = 1$, and $\delta = 1$. The average value of the entropy H is about 5.5 for the examples in Güttler et al. and hence the average deviation from C_{opt} is in this example at least 18% for the code constructed by Cot.

IV. Conclusion

A new algorithm for constructing nearly optimal prefix codes in the case of unequal probabilities and unequal letter costs has been described. A theoretical estimate of the cost of the constructed code has been given. Numerical examples suggest that the algorithm is superior to previously suggested approximation algorithms. The algorithm is very efficient in its time and space requirements.

Bibliography

[1] Krause, R.M. : "Channels which transmit letters of unequal duration", Inf. and Control 5, pp. 13-24, 1962

[2] Csiszar : "Simple proofs of some theorems on noiseless channels", Inf. and Control 14, pp. 285-298, 1969

[3] Altenkamp & Mehlhorn : "Codes : Unequal Probabilities, Unequal Letter Costs", to appear JACM

[4] Cot : "Characterization and Design of Optimal Prefix Codes", Ph.D. Thesis, Stanford University, June 1977

[5] Shannon, C.E. : "A mathematical theory of communication", Bell Systems Techn. J., 27, pp. 379-423, 623-656, 1948

[6] Bayer : "Improved Bounds on the Cost of Optimal and Balanced Binary Search Trees, MIT, Project MAC, Technical Report

[7] Güttler, Mehlhorn, Schneider, Wernet : "Binary Search trees : Average and worst case behaviour", (GI-Jahrestagung 1976), Informatik-Fachberichte 5, Springer-Verlag, 301-313.