An Efficient Algorithm for Constructing Nearly Optimal Prefix Codes

KURT MEHLHORN

Abstract—A new algorithm is presented for constructing nearly optimal prefix codes in the case of unequal letter costs and unequal probabilities. 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. A set \(p_1, p_2, \ldots, p_n\) of probabilities are given, with \(p_i \geq 0\) \(\sum_{i=1}^{n} p_i = 1\) and a set \(a_1, a_2, \ldots, a_i\) of letters; the letter \(a_i\) has cost \(c_i \in \mathbb{R}\), where \(c_i > 0\). A prefix code \(T\) over the alphabet \(\Sigma = \{a_1, a_2, \ldots, a_i\}\) is a set \(U_1, \ldots, U_n\) of words in \(\Sigma^*\) such that no \(U_i\) is a prefix of any \(U_j\) for \(i \neq j\). Let

\[ U_i = a_{i_1} a_{i_2} \cdots a_{i_t} \]

be the \(i\)th codeword. Its cost \(C(U_i)\) is defined as the sum of the letter costs, i.e.

\[ C(U_i) = c_{i_1} + c_{i_2} + \cdots + c_{i_t}. \]

Finally, the average cost of the code \(T\) is defined to be

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

At present there is no efficient algorithm for constructing an optimal (equal to minimum average cost) code given \(p_1, \ldots, p_n\) and \(c_1, \ldots, c_n\). Karp [1] formulated the problem as an integer programming problem, and hence his algorithm may have exponential time complexity. Various approximate algorithms are described in the literature (Krause [2], Csiszár [3], Altenkamp and Mehlhorn [4], Cot [5]). They construct codes \(T\) such that

\[ H(p_1, \ldots, p_n) \leq C_{opt} \leq C(T), \]

where \(H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i\) is the entropy of the probability distribution, \(c\) is defined so that \(\sum_{i=1}^{n} 2^{-c_i} = 1\) (a root of the characteristic equation of the letter costs), \(C_{opt}\) is the cost of an optimal code, \(f(c_1, \ldots, c_i)\) is some function of the letter costs, and \(\gamma\) is a small constant. In most cases (Krause [2], Csiszár [3], Altenkamp and Mehlhorn [4]) \(f(c_1, \ldots, c_i) = \max \{c_i | 1 \leq i \leq t\}\), while for Cot [5] \(f(c_i, \ldots, c_i)\) is a more complex function.

Here we describe another approximate algorithm and derive 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 \(O(t\cdot n)\) 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. At the first node of the code tree we split the set of given probabilities into two parts of probabilities \(p\) and \(1-p\), respectively (Fig. 1). 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

\[ G(p) = \frac{-p \log p - (1-p) \log (1-p)}{(-p \log 2^{-c_1} - (1-p) \log 2^{-c_2}) \cdot \frac{1}{c}}, \]

for all \(c \neq 0\).

It is easy (by elementary calculus) to see that \(G(p)\) is maximal for \(p = 2^{-c_1}\) and \(1-p = 2^{-c_2}\), where \(c\) is chosen so that \(2^{-c_1} + 2^{-c_2} = 1\). Hence \(G(p) \leq c\) for all \(p\) and \(G(2^{-c_1}) = c\).

The previous argument suggests the following approximate algorithm. Try to split the given set of probabilities into two parts of probabilities \(p\) and \((1-p)\) respectively so as to make \(p - 2^{-c_1}\) 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. We are given probabilities \(p_1, p_2, \ldots, p_6\) = (0.3, 0.1, 0.05, 0.25, 0.2, 0.1) and a code alphabet \(a_1, a_2\) with costs \((c_1, c_2) = (1, 2)\). We choose \(c\) so that \(2^{-c_1} + 2^{-c_2} = 1\). Then \(2^{-c_1} = 0.618\).

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 \(2^{-c_1}\) and \(2^{-c_2}\), respectively (Fig. 2). The split goes through the right half of \(p_4\), so we assign the letter \(a_1\) to \(p_1, p_2, p_3\), and \(p_4\) and the letter \(a_2\) to \(p_5, p_6\) (Fig. 3). 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).

\textbf{Caution:} At this point our approach differs from the one taken by Krause, Csizár, and Altenkamp and Mehlhorn. After having split the unit interval into two parts in the first step, they split the interval of length $2^{-cc_1}$ in the ratio $2^{-cc_1}$ to $2^{-cc_2}$ 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.

We proceed with our example. In Fig. 4 the split goes through the right half of $p_3$, so we assign the letter $a_1$ to $p_1, p_2, p_3$ and the letter $a_2$ to $p_4$ (Fig. 5). Proceeding in this fashion the code is constructed in Fig. 6. This code has cost

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

So much for the intuitive description of the algorithm. For the precise definition by a pseudo-algorithmic language (ALGOL) program we need some notation. Let $c \in \mathbb{R}$ be such that $\sum_{j=1}^{n} 2^{-cc_j} = 1$. Then $2^{-c}$ is traditionally called the root of the characteristic equation of the letter costs. Let $P_k = p_1 + p_2 + \cdots + p_k$, $0 < k < n$, and $s_k = p_1 + p_2 + \cdots + p_{k-1} + p_k/2$, $1 \leq k < n$.

The command \texttt{CODE} $(1, n, \epsilon)$ constructs a prefix code for the probability distribution $p_i, \cdots, p_n$. Here $\epsilon$ denotes the empty word over the alphabet $(a_1, \cdots, a_{\ell})$.

\begin{verbatim}
procedure CODE (l, r, U):
comment: l and r are integers, $1 \leq l < r \leq n$, and $U$ is a word over $(a_1, \cdots, a_{\ell})$. We will construct codewords for $p_l, p_{l+1}, \cdots, p_r$. The word $U$ is a common prefix of codewords $U_i, U_{i+1}, \cdots, U_r$.
begin
if $l = r$
then
begin
we take $U$ as the codeword $U_i$
else begin $L \leftarrow P_{l-1}$; $R \leftarrow P_r$
for $m, 1 \leq m \leq l$
do
begin
$L_m \leftarrow L + (R - L) \cdot \sum_{j=1}^{m-1} 2^{-cc_j}$;
$R_m \leftarrow L_m + (R - L) \cdot 2^{-cc_m}$;
$I_m \leftarrow \{i; L_m < s_i < R_m\}$
end;
end
Comment: $I_m, 1 \leq m \leq l$, is a (not necessarily nontrivial) partition of the set $(l, \cdots, r)$. Since we certainly do not want to assign the same letter to all probabilities $p_i, \cdots, p_r$, we need to make sure that the partition is nontrivial. The easiest way to ensure nontriviality is to force the use of letters $a_1$ and $a_2$, i.e. to make $I_l$ and $I_r$ nonempty;

if $I_l = \emptyset$
then begin let $m$ be minimal with $I_m \neq \emptyset$
$I_l \leftarrow \{\}$; $I_m \leftarrow I_m - \{\}$
end;
if $I_r = \emptyset$
then begin let $m$ be maximal with $I_m \neq \emptyset$
$I_r \leftarrow \{\}$; $I_m \leftarrow I_m - \{\}$
end;
\end{verbatim}
Fig. 7. First two steps of fractional expansion method.

partition as it exists at this point of the program;
for \( m = 1 \leq m \leq t \)
do
if \( I_m \neq \emptyset \) then CODE (min \( I_m \), max \( I_m \), \( Ua_m \));
end.
end.

Remark: Procedure CODE is a generalization of Shannon's binary splitting algorithm [6] for constructing nearly optimal codes over a binary alphabet. It has been generalized in a different direction in the past by Krause, Csiszar, and Altenkamp and Mehlhorn, who view the binary splitting algorithm as a fractional expansion process.

Consider the binary fraction \( 0.x_1x_2\cdots x_m \) with \( x_i \in \{0,1\} \). We can define the real number represented by that binary fraction recursively by

\[
\text{Num}(x_m) = \begin{cases} 0 & \text{if } x_m = 0 \\ 1 & \text{else} \end{cases}
\]
\[
\text{Num}(x_i x_{i+1} \cdots x_m) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1/2 + \frac{1}{2} \text{Num}(x_{i+1} \cdots x_m) & \text{else} \end{cases}
\]

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 \( 2^{-c_0}:1-2^{-c_0} \). Then we should define Num as follows.

\[
\text{Strangenum}(x_m) = \begin{cases} 0 & \text{if } x_m = 0 \\ 2^{-c_0} & \text{else} \end{cases}
\]
\[
\text{Strangenum}(x_i x_{i+1} \cdots x_m) = \begin{cases} 0 & \text{if } x_i = 0 \\ 2^{-c_0} \cdot \text{Strangenum}(x_{i+1} \cdots x_m) & \text{else} \end{cases}
\]

We are now ready to take up the above remark (labeled caution) and to outline the fractional expansion approach of our example. Consider the fractional expansions of the 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 \( s_6 \). Fig. 7 in addition shows the second digits in the expansion of \( s_1, s_2, s_3, s_4 \). 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 construct the code shown in Fig. 8, of cost 3.75.

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, when \( t = 2 \), \( c_1 = c_2 = 1 \). It generally yields much better codes (cf. the experimental results at the end of the paper).

Theorem: Given probabilities \( p_1, \ldots, p_n \) and letters \( a_1, \ldots, a_n \) of cost \( c_1, \ldots, c_n \) and a real number \( c \) such that \( \sum_{m=1}^n 2^{-c_m} = 1 \), procedure CODE constructs a code tree \( T \) of average cost \( C(T) \) with

\[
c \cdot C(T) < H(p_1, \ldots, p_n) + 1 - p_1 - p_n + c c_{\text{max}}.
\]

where \( c_{\text{max}} = \max \{c_m; 1 \leq m \leq t\} \).

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

Procedure CODE constructs a code tree \( T \) for probabilities \( p_1, \ldots, p_n \). Let \( v \) be any node of the complete infinite tree over the letters \( a_1, \ldots, a_n \), 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

\[
w(v) = \sum p_i; \ U \text{ is a prefix of codeword } U_i \text{ for } p_i
\]
and

\[
w_m(v) = w(v_m),
\]
where \( v_m \) corresponds to \( Ua_m \). Then

\[
w(v) = w_1(v) + w_2(v) + \cdots + w_r(v).
\]

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

\[
w(v) = p_l + p_{l+1} + \cdots + p_r.
\]

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

Lemma 1:
1) The cost \( C(T) \) of the code tree \( T \) is

\[
C(T) = \sum_{v \in N_T} c_j w(j).
\]

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

\[
H(p_1, \ldots, p_n) = \sum_{v \in N_T} w(v) \cdot H\left(\frac{w_1(v)}{w(v)}, \ldots, \frac{w_r(v)}{w(v)}\right).
\]

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

\[
C(T) = \sum_{i=1}^n p_i \cdot \text{cost}(U_i)
\]
we sum over the leaves of the code tree. If for every interior node \( v \) and letter \( q \), we consider those codewords \( U_j \) that go through \( v \) and use the letter \( q \) in node \( v \) then we obtain the summation formula given in the lemma.

Lemma 1 allows us to write

\[
c \cdot C(T) - H(p_1, \cdots, p_n) = \sum_{v \in N_T} \left[ \frac{c}{m} w_m(v) - w(v) \cdot H \left( \frac{w_1(v)}{w(v)}, \cdots, \frac{w_l(v)}{w(v)} \right) \right] = \sum_{v \in N_T} \left[ \frac{c}{m} w_m(v) \left( \log 2^{c_m} + \log \frac{w_m(v)}{w(v)} \right) \right]
\]

(1)

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

\[
E(v, m) := \frac{w_m(v)}{w(v)} \left( \log 2^{c_m} + \log \frac{w_m(v)}{w(v)} \right).
\]

(2)

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

**Lemma 2**: Consider any call CODE \((i, r, U)\), let node \( v \) correspond to the word \( U \) and suppose \( I < r \). Let sets \( I_1, \cdots, I_m \) be defined as in Procedure CODE. Then, for \( 1 \leq m < r \),

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

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

c) if \( |I_m| > 2 \), and \( e = \min I_m \), \( f = \max I_m \), then, for \( 2 < m < r \),

\[
\frac{w_m(v)}{w(v)} < 2^{-c_m} + \frac{p_e + p_f}{2 \cdot w(v)} < 2 \cdot 2^{-c_m}
\]

also

\[
\frac{w_1(v)}{w(v)} < 2^{-c_1} + \frac{p_f}{2w(v)} < 2 \cdot 2^{-c_1}
\]

\[
\frac{w_l(v)}{w(v)} < 2^{-c_l} + \frac{p_e}{2w(v)} < 2 \cdot 2^{-c_l}
\]

**Proof**: a) and b) are obvious. Consider c). Suppose first \( 2 < m < r \). Fig. 9 shows the meaning of \( e \) and \( f \). Then \( w_m(v) = p_e + p_{e+1} + \cdots + p_{f-1} + p_f \) and \( p_e/2 + p_{e+1} + \cdots + p_{f-1} + p_f/2 < 2^{-c_m} \cdot w(v) \) by definition of \( w(v) \), \( w_m(v) \) and \( I_m \). Hence

\[
w_m(v) < 2^{-c_m} \cdot w(v) \cdot 2^{-c_m} \cdot w(v) \cdot 2^{-c_m} \cdot w(v) \cdot 2^{-c_m} \cdot w(v) \]

If \( m = 1 \) we even have

\[
p_e + p_{e+1} + \cdots + p_{f-1} + p_f / 2 < 2^{-c_1} \cdot w(v)
\]

and hence

\[
w_1(v) < 2^{-c_1} \cdot w(v) < p_f / 2 < 2^{-c_1} \cdot w(v).
\]

An analogous statement holds for \( m = t \).

We are now ready to derive the upper bound on \( E(v, m) \) defined in (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 \( w_m(v)/w(v) < 1 \).

Hence

\[
E(v, m) < \frac{w_m(v)}{w(v)} \cdot \log 2^{c_m} = (c_m \cdot p_e) / w(v).
\]

**Case c**: \( |I_m| > 2 \). Let \( e = \min I_m \), \( f = \max I_m \). Let \( y := 2^{-c_m} \) and \( x := w_m(v)/w(v) - 2^{-c_m} \). Then \( x < (p_e + p_f)/2w(v) < 2^{-c_m} \) by Lemma 2. We may rewrite \( E(v, m) \) as

\[
E(v, m) = (x + y) \cdot \log 1/y + \log(x + y) \cdot \log(1 + x/y).
\]

**Lemma 3**: Let \( 0 < x < y \) and \( 0 < y \). Then

\[
(x + y) \cdot \log(1 + x/y) < 2x.
\]

**Proof**: Consider

\[
f(x) = 2x - (x + y) \cdot \log(1 + x/y).
\]

Then

\[
f'(x) = 2 - \log(1 + x/y) - \frac{(x + y) / y}{\ln 2 - (1 + x/y)}
\]

\[
= (2 - 1/\ln 2) - \log(1 + x/y).
\]

Thus \( f' \) is monotonically decreasing and hence \( \min(f(x); 0 < x < y) = \min(f(0), f(y)) = 0 \).

From Lemma 3 we conclude that

\[
E(v, m) < 2x = (p_e + p_f) / w(v);
\]

for \( m = 1 \) we can even conclude \( E(v, m) < p_e / w(v) \) and for \( m = t \), \( E(v, m) < p_r / w(v) \). In every case we now have an upper bound on \( E(v, m) \).

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

In summary, we use each probability \( p_i \) at most once in a bound of type b) and at most once in a bound of type c). Furthermore the argument above shows that \( p_1 \) and \( p_n \) are never used in a bound of type c).
We will now substitute the bounds on $E(v,m)$ into (1), our expression for the difference $c \cdot C(T) - H(p_1, \cdots, p_n)$. The bounds of type b) contribute at most $c \cdot c_{\text{max}} \cdot \sum_{i=1}^{n} p_i = c \cdot c_{\text{max}}$, where $c_{\text{max}} = \max\{c_m; 1 < m < t\}$ and the bounds of type c) contribute at most $\sum_{i=1}^{n} = 1 - p_1 - p_n$. Hence

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

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

III. IMPLEMENTATION AND EXPERIMENTAL DATA

Altenkamp and Mehrlhorn [4] 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 [4] for details.

In Gütter et al. [8] the algorithms described in [4] (which are very similar to the one described by Krause [2] and Csiszár [3]) and the algorithm described here were compared in the binary case with equal letter costs, $t = 2$ and $c_1 = c_2 = 1$. Two hundred examples were run; for each of them the optimal code was constructed. Fig. 10 shows the average and maximal values of $C_1 / C_{\text{opt}} \cdot 100$ and $C_2 / C_{\text{opt}} \cdot 100$ where $C_{\text{opt}}$ 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 Mehrlhorn, respectively.

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

$$H(p_1, \cdots, p_n) / c + \delta < C < H(p_1, \cdots, p_n) / c + \delta + c_{\text{min}}$$

where

$$\delta = \sum_{i=2}^{c} c_i \log_2 \left( \lambda_i / \lambda_{i-1} \right)$$

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

for $1 < i < t$. He does not describe a detailed 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 Gütter et al. and hence the average deviation from $C_{\text{opt}}$ is in this example at least 18 percent 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 that algorithm is superior to previously suggested approximation algorithms. The algorithm is very efficient in its time and space requirements.

REFERENCES